# DISASTER RESILIENT MESH NETWORK WITH DATA SYNCHRONIZATION USING NERVENET

**LIM WEI SEAN**

**A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering**

**Lee Kong China Faculty of Engineering and Science**
**Universiti Tunku Abdul Rahman**

**April 2021**

# DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

| | | |
|---|---|---|
| Signature | : | [illegible] |
| Name | : | Lim Wei Sean |
| ID No. | : | 1702106 |
| Date | : | 9 April 2021 |

## APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DISASTER RESILIENT MESH NETWORK WITH DATA SYNCHRONIZATION USING NERVENET"** was prepared by **LIM WEI SEAN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

| | | |
|---|---|---|
| Signature | : | [illegible] |
| Supervisor | : | Tham Mau Luen |
| Date | : | 3/5/2021 |

| | | |
|---|---|---|
| Signature | : | [illegible] |
| Co-Supervisor | : | Dr. Lee Ying Loong |
| Date | : | 3/5/2021 |

# ACKNOWLEDGEMENTS

This fruitful project can only be accomplished with the advice and guidance from my supervisor, Ir Ts Dr Tham Mau Luen and my co-supervisor, Dr Lee Ying Loong. Their dedicated bits of advice have provided me with many insights while making challenging decisions throughout the project. I would like to express my gratitude for what they have contributed to the project and what have done for me.

Additionally, I would like to convey my sincere gratitude to Dr Owada Yasunori from Japan’s National Institute of Communication and Information Technology for providing pieces of advice and information regarding NerveNet. His clear guides and documents regarding NerveNet provided by him and his team have led the project through like a breeze.

Furthermore, I would like to thank my supportive parents and elder brother that cares for me through the research project. Their support keeps my spirits up for all the difficult times throughout the project.

Finally, I would also like to show my appreciation to all my supportive friends and coursemates. Their support and presence themselves have continuously given me inspiration and motivation that led to the success of this project.

## ABSTRACT

Natural disasters occur frequently around the world. Internet of Things (IoT) sensors such as video cameras can detect such cataclysmic events, track the number of victims and subsequently initiate rescue actions. How to disseminate the critical information, however, remains an open issue especially when there are communication breakdowns. This project aims to develop a regional disaster response platform using NerveNet, which is a mesh networking technology provided by Japan NICT. By utilising NerveNet Hearsay daemon, images can be wirelessly synchronized in multiple NerveNet nodes' database. To facilitate the emergency management, a cloud monitoring dashboard to visualize multiple regional response and monitoring networks has been designed and developed. Serving as a proof of concept, a NerveNet testbed consisting of two base stations and one gateway has been implemented. Experimental results validate the feasibility of the proposed platform from two perspectives, namely network and data synchronization performance. The former measures throughput, delay, and jitter, whereas the latter focuses on analysing the latency of image synchronization. The project findings can serve as the guideline for designing a disaster response and monitoring platform in not only Malaysia but also other ASEAN countries.

# TABLE OF CONTENTS

## LIST OF TABLES

# LIST OF FIGURES

## LIST OF SYMBOLS / ABBREVIATIONS

| | |
|---|---|
| PGW | Packet Data Network Gateway |
| API | Application Programming Interface |
| DBMS | Database Management System |
| LAN | Local Area Network |
| WAN | Wide Area Network |
| CAN | Controller Area Network |
| PROFIBUS | Process Field Bus |
| WLAN | Wireless Local Area Network |
| IoT | Internet of Things |
| WMN | Wireless Mesh Network |
| AODV | Ad Hoc On-Demand Distance Vector |
| OLSR | Optimized Link State Routing Protocol |
| SQL | Relational Database |
| MySQL | Non-Relational Database |
| JSON | JavaScript Object Notation |
| IAM | Identity and Access Management |
| CLI | Command Line Interface |
| IP | Internet Protocol |
| DHCP | Dynamic Host Configuration Protocol |
| OMG | Originator Message |
| NWGN | New Generation Network |
| NICT | National Institute of Information and Communications Technology, Japan |
| BS | Base Station |
| GW | Gateway |
| NM | Network Manager |
| AP | Access Point |
| OS | Operating System |
| XML | Extensible Markup Language |
| PDR | Packet Delivery Ratio |
| RTT | Round-Trip Time |
| PDV | Packet Delay Variation |

| | |
|---|---|
| VoIP | Voice Over Internet Protocol |
| OSI | Open Systems Interconnections |
| DRY | “Don’t Repeat Yourself” |
| RAM | Random-Access Memory |
| AWS | Amazon Web Services |
| GCP | Google Cloud Platform |
| SSH | Secure Shell |
| SCP | Secure Copy Protocol |
| SFTP | Secure File Transfer Protocol |
| DTDL | Digital Twin Definition Language |
| HTML | Hypertext Markup Language |
| CSS | Cascading Style Sheets |
| JS | JavaScript |
| HTTP | Hypertext Transfer Protocol |
| MQTT | Message Queuing Telemetry Transport |
| WAP | Wireless Access Point |
| ERB | Ethernet Remote Bridge |
| STA | Station |
| REST | Representational State Transfer |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| ICMP | Internet Control Message Protocol |
| MTU | Maximum Transmission Unit |
| UAV | Unmanned Aerial Vehicle |

## LIST OF APPENDICES

# CHAPTER 1

# INTRODUCTION

## 1.1 General Introduction

The resilience of a network is always an issue in deploying a fault-tolerant network. In places like the United States, China and Indonesia where there is a high risk of natural disasters, a resilient network consisting of various sensor devices is required to provide crucial information during a disaster strike. On March 11, 2011, the East Japan Great Earthquake has damaged approximately 29000 cellular towers (DoCoMo, 2011). These damages have restricted families and friends to communicate, halted the broadcast of evacuation notice and most importantly prevented collections of recent and historical information for the rescue teams.

This problem cannot be avoided by the current tree topology communication architecture. The network will not be functioning from the lower part of the hierarchy if a higher component or service has a fault or malfunction. This makes our current network architecture vulnerable to faults. For example, in the 4G LTE architecture, if the Packet Data Network Gateway (PGW) is damaged, the area connectivity to the Internet will be affected. This causes trouble for disaster victims to contact their families or rescue authorities and also to rescue authorities to retrieve data about the affected areas.

To overcome this problem, a mesh network can be used rather than a tree topology network. A mesh network is a network topology that allows devices to be connected directly and allows re-transmission of data packets to other members of the network. Mesh topology can be utilized to create a resilient network as it can transmit data through multiple routes.

During a disaster, information on the condition and number of victims in the disaster area is crucial for the rescue team to execute their jobs. Evacuation notice broadcast is also important to minimize the damage caused by a disaster. However, a network that allows victims to communicate with the community service and rescue applications directly without connecting to the Internet does not currently exist in Malaysia. Fortunately, there is a solution for a disaster-resilient mesh-topological network called NerveNet, which is provided by Japan NICT. In NerveNet, each node is independent of the other and it is tolerant to link disconnections and system failures because of its mesh structure and fast route switching on layer 2.

Information availability during a disaster can be provided by either implementing an application-specific Application Programming Interface (API) or synchronizing the application database. However, it will be troublesome to create an application-specific API just to have information in multiple locations as it would require modifying the code of the publishing node to add a broadcast destination and require the receiving node to have the API to receive the data. Rather than pushing the data directly from the application, it is much easier to synchronize the application database passively. This can be done by using the database replication provided in most of the modern Database Management Systems (DBMS) such as MySQL and MongoDB. In NerveNet, it can be simply done using the Hearsay Protocol.

## 1.2 Importance of the Study

Although Malaysia is located in a geologically stable region, Malaysia is still often vulnerable to floods and other disasters. The damages caused by the floods can be devastating as seen in the 2014-15 Malaysian flood. This shows the importance to have a resilient network for emergency use in Malaysia. For the emergency response authorities to easily broadcast emergency or evacuation notices to a public notice board, vending machines and smart advertisement boards, a broadcasting mechanism must be incorporated in the regional network. This can be achieved by having a database synchronization feature in the regional network to synchronize the databases of the regional emergency response authorities with the notice application databases. Database synchronization feature is also used for data collection of regional data by emergency response authorities. The data collection allows the regional emergency response authorities to monitor the conditions of the region and provide guides and notice for evacuation during pre- and post-disaster. To allow nationwide monitoring and control, a cloud monitoring and control dashboard is essential to visualize and provide signals for multiple regional mesh networks.

## 1.3 Problem Statement

The absence of a regional resilient network topology and architecture for emergency use is one of the shortcomings of the current network in Malaysia. With a resilient network, a database synchronization feature must also be set in place in the network. The database synchronization feature allows applications to pull data from each other natively in the regional resilient network. A cloud-based monitoring dashboard is also required to obtain insights from multiple regional networks and to provide nationwide responses.

## 1.4 Aim and Objectives

This study aims to design and develop a NerveNet mesh network for regional disaster response applications. The objectives of this study are:

(i) To deploy a NerveNet mesh network testbed with data synchronization
(ii) To design and deploy a cloud monitoring dashboard for the regional disaster monitoring network
(iii) To evaluate the performance of the NerveNet testbed in terms of its network and database synchronization efficiency

## 1.5 Scope and Limitation of the Study

Since this project is in collaboration with Japan NICT under the funding of ASEAN IVO, the focus is to deploy the NerveNet for regional disaster response applications. The deployment, however, can be done only on a small-scale testbed due to the ongoing pandemic issue. Besides that, the project development progress was affected by the delay of technical support from the NICT side such as incomplete documentations of NerveNet OS installation. Nevertheless, the small-scale testbed has been successfully implemented and thorough analysis has been conducted.

## 1.6 Contribution of the Study

A three-node NerveNet mesh network testbed is deployed and worked as a network of disaster monitoring cluster in this study. Data between the disaster monitoring base stations are synchronized within the mesh network and to the cloud for data resiliency. A cloud monitoring dashboard has been designed, developed and deployed for monitoring and visualization of disaster and network situations in multiple NerveNet networks. Since this project is funded by NICT ASEAN IVO, the outcome of this case study will serve as the guideline for NerveNet deployment in other countries such as Thailand and Myanmar.

## 1.7 Outline of the Report

This report consists of five major chapters. The first chapter provides an overview of the project's topic, aim and objectives. Chapter 2 uncovers some computer networking and web development knowledge related to this project. Chapter 3 includes the detailed work schedule and methodology of the project. Chapter 4 shows the implementation of the deployed system as well as the result of a network performance analysis on the system. Some remarks, future recommendations and possible enhancements are covered in the final chapter.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Introduction

A computer network is a cluster of connected computers. The connection can be either wired together in terms of copper wires and fiber optics or connected wirelessly via Wi-Fi and radio waves (Sandberg, 2015). There are two kinds of networks, Local Area Network (LAN) and Wide Area Network (WAN). LAN allows devices within an organization to communicate with each other. On the other hand, WAN allows devices from different organizations to communicate with each other. WAN covers a wider area which usually spans across one or multiple regions and even continents.

Network resilience is defined as "the ability of a network to defend against and maintain an acceptable level of service in the presence of such challenges" (Smith et al., 2011). According to Sterbenz et al. (2010), the resilience of a network is enabled by seven principles which are:

(i) Self-protection and security
(ii) Connectivity and association
(iii) Redundancy
(iv) Diversity
(v) Multilevel resilience
(vi) Context Awareness
(vii) Translucency

Self-protection and security are the capabilities of the network to protect itself from challenges. Connectivity and association are the capabilities of the network to continue the communication even without a stable end-to-end connection between the devices. Redundancy can be achieved by having replication of entities in the network. In terms of information redundancy, copies of the data are stored in multiple network nodes to provide fault-tolerant data availability. Diversity focuses on providing alternatives to avoid operations within the network to degrade. Multilevel resilience enables resilient features at different protocol layers. Context-awareness is the network feature to sense the network condition and provide remedial actions. Translucency is the visibility between layers against the degree of abstraction within the network. With most of the seven principles enabled in the network, the network can be considered resilient, but it also depends on the possible risk faced by the network.

A disaster tolerant network utilizes distributed applications and resources on a resilient network to provide emergency and communications services during and after a disaster. These networks work independently to prevent network congestions, especially at the network core level. It provides local applications for emergency response and community service applications such as data collection, rescue requests and emergency notice broadcastings. As disaster cannot be accurately predicted, a data synchronization feature is required in these networks provided by any protocol layer to have a constant update of the regional condition.

## 2.2 Network Topology

Network topology has a direct impact on network resilience. There are four basic kinds of network topologies in computer networking which are point-to-point topology, bus topology, ring topology and star topology. Additionally, two advanced network topologies namely mesh topology and tree topology can be formed based on the extension of point-to-point topology and star topology, respectively.

### 2.2.1 Point-to-Point Topology Network

Figure 2.1 shows a point-to-point topology network where two devices connect directly with each other through a wired or wireless communication interface. An example of point-to-point communication is Bluetooth and Wi-Fi ad-hoc mode.

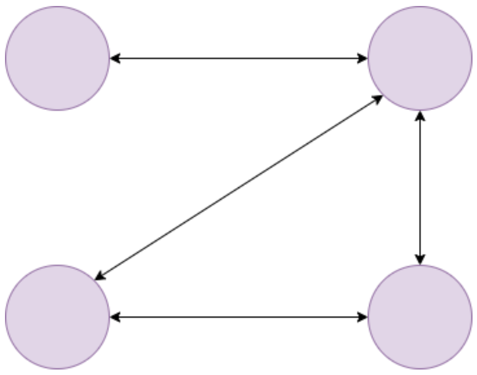


Figure 2.1: Point-to-Point Topology.

### 2.2.2 Bus Topology Network

Figure 2.2 shows a bus topology network that connects all devices using one single cable. Bus topology networks only allow unidirectional communications. This network topology is seldom used in computer networking nowadays, but it is popular in industrial device communication. Controller Area Network (CAN) bus protocol, Modbus protocol and Process Field Bus (PROFIBUS) protocol are some of the communication protocols that use this network topology.

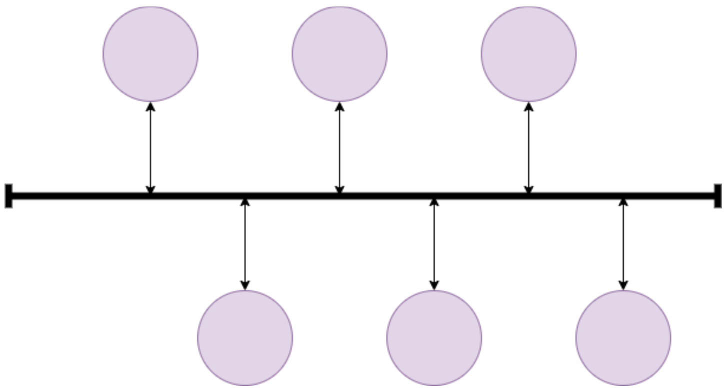


Figure 2.2: Bus Topology.

### 2.2.3 Ring Topology Network

Devices in a ring topology network connect to two neighbouring devices forming ring-shaped connectivity. This allows devices within the network to communicate with each other without direct connection. Ring topology network is a subset of partial mesh topology networks. If the hub of a star topology network is connected with other hubs in a ring topology fashion, it is called a token ring topology network. MOXA’s Turbo Ring redundancy protocol is a protocol that utilizes a ring topology. A ring topology is shown in Figure 2.3.

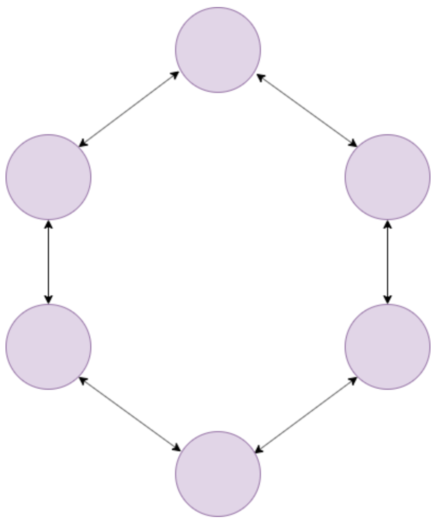


Figure 2.3: Ring Topology.

### 2.2.4 Star Topology Network

Star topology is the most common network topology used in computer networking. Star topology network requires a central hub to connect all the devices. The central hub relays the data packet from one device to the other based on the destination of the data packet. As it is very dependent on the central hub, failure on the hub will cause the whole network to fail. A common use of this topology is the wireless access point (AP) which forms the Wireless LAN (WLAN) of most organizations. An example of a star topology network is shown in Figure 2.4.

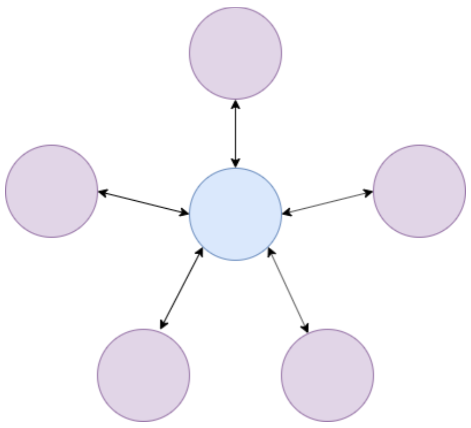


Figure 2.4: Star Topology.

### 2.2.5 Mesh Topology Network

Figure 2.5 depicts a mesh topology network where all devices are connected and are able to relay data packets from one device to the other. If all devices are directly connected, this kind of mesh network is called a full mesh network. On the other hand, if not all devices are directly connected but still able to communicate through relaying of data packets, this network is called a partial mesh network. Mesh network is very advantageous in terms of the resilience of the network as there is still an alternative routing path to route its packets even if an intermediate node has failed in the original path. However, a mesh network is often much more difficult to set up. Mesh networks are currently common for home automation, Internet of Things (IoT) and computer networking in remote areas.

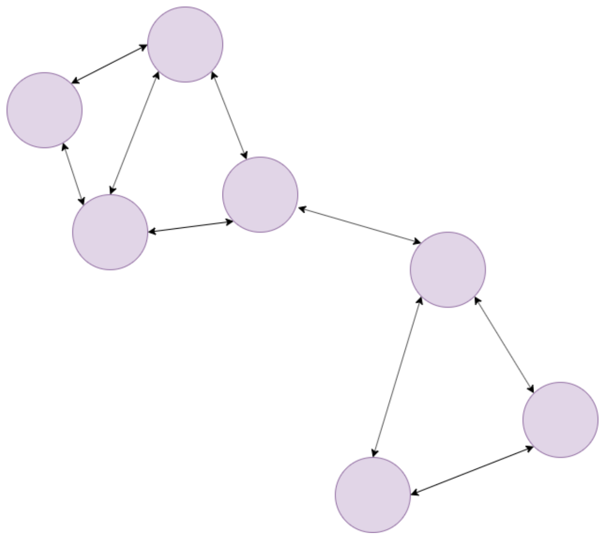


Figure 2.5: Mesh Topology.

### 2.2.6 Tree Topology Network / Extended Star Network

When the hub of the star topology networks is connected to another hub, this forms an extended star network which is also called a tree topology network due to its tree-like hierarchy architecture. Tree topology networks are very suitable for larger networks as they can be segregated into multiple branches for simple management. However, the main disadvantage of a tree topology is that it is very dependent on the central hub of the network. If there is any failure on any higher-level hubs, all its child networks cannot communicate with each other and any network above the failed hub’s network. The hubs are said to be the single-point failure of the network. The use of this topology is to segregate large networks into smaller star networks. This network can be set up using network devices such as network switches and routers. An example of a tree topology is shown in Figure 2.6.

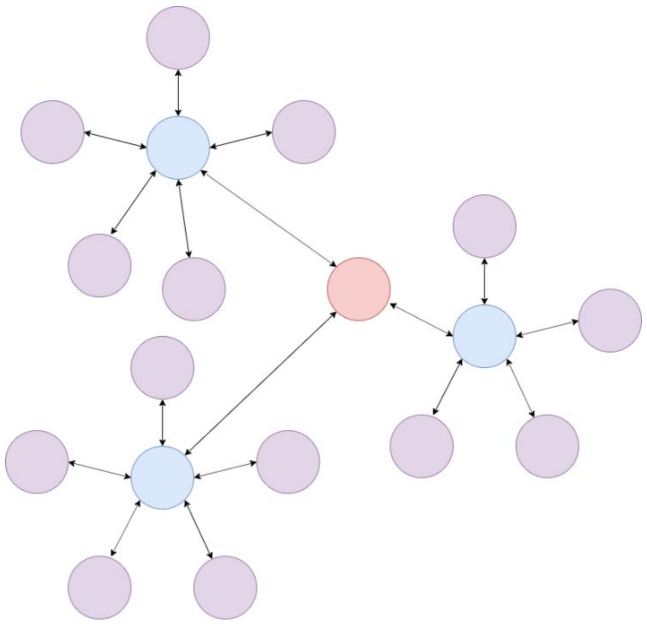


Figure 2.6: Tree Topology.

### 2.2.7 Hybrid Topology Network

When various network topologies are joined together to form a larger network, this network forms a hybrid network. Figure 2.7 shows a hybrid topology network of a mesh network and star network.

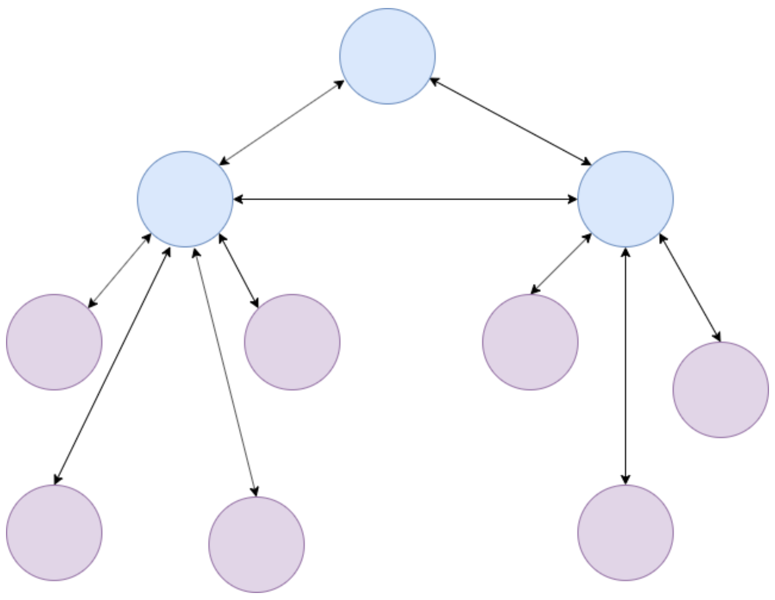


Figure 2.7: Hybrid Topology of Mesh Topology and Star Topology.

## 2.3 Wireless Mesh Network (WMN)

Wireless Mesh Network (WMN) is a network where devices are connected wirelessly in a mesh topology. This type of mesh network is different as wireless interfaces have more considerations compared to wired mesh networks. As wireless medium tends to introduce more interference and there is no physical connection, it is a challenge to detect a failure at other nodes. Special routing protocols are developed to manage package routing, member discovery and other challenges of a wireless mesh network. Some wireless mesh routing protocols are BATMAN, NerveNet, Ad hoc On-Demand Distance Vector (AODV) and Optimized Link State Routing Protocol (OLSR).

## 2.4 Database

A computer database is a collection of digital data and is manageable through software called a Database Management System (DBMS). There are many distributions of databases, but they can be generally divided into two major categories which are relational database (SQL) and non-relational database (NoSQL).

### 2.4.1 Relational Database (SQL)

Relational databases store their data as rows in a table. A SQL DBMS can hold multiple databases. Each database will hold multiple tables and each row in the table represents a data entry. Fields of the data are represented as columns in the SQL table. This can be illustrated in the stacked Venn diagram as shown in Figure 2.8. Examples of SQL databases are MySQL, PostgreSQL and SQLite.

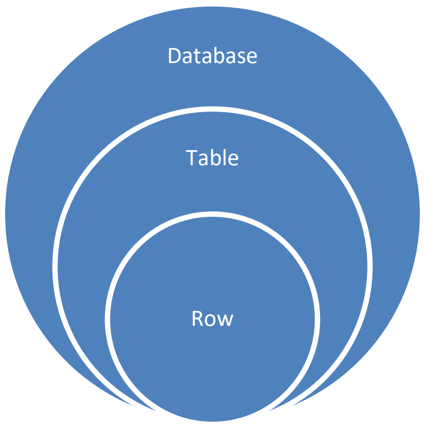


Figure 2.8: Stacked Venn Diagram of a Typical SQL Database Architecture.

### 2.4.2 Non-relational Database (NoSQL)

NoSQL databases store data in a form that it does not require to define the data structure first and sometimes could not do so either. There are many types of NoSQL databases which include Document store databases, Key-value store databases, Column-oriented databases and graph databases. NoSQL DBMS can usually hold multiple databases. The unique features of NoSQL databases simplify many data storage problems and render big data analysis possible. As NoSQL is still considerably new, there is no standardized query language. Thus, this makes migration between different databases much more challenging (SQL Vs NoSQL Exact Differences And Know When To Use NoSQL And SQL, 2021).

#### 2.4.2.1 Document Store Database

Document store NoSQL databases store their data as documents in a collection. Each database in its DBMS can hold multiple collections and each document represents the data in the collection. Documents are data that have a JavaScript Object Notation (JSON) like format (Drake, 2019). This can be illustrated in the stacked Venn diagram as shown in Figure 2.9. MongoDB, CouchDB and Azure Cosmos DB are three common Document store NoSQL databases.

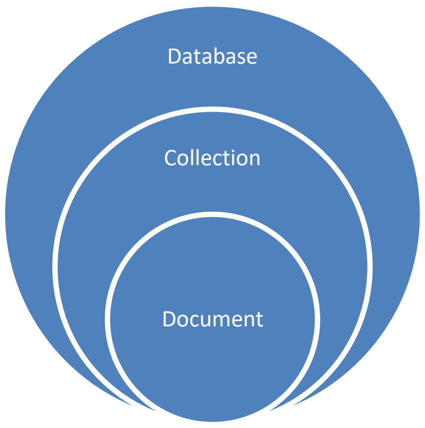


Figure 2.9: Stacked Venn Diagram of a Typical Document Store NoSQL Database Architecture.

#### 2.4.2.2 Key-Value Store Database

Key-value store NoSQL databases are similar to document store databases. However, it has only a single collection and only stores its values as an opaque blob. This type of data storage model is usually used for non-persistent storage such as caching, queueing and session storage due to its simplicity, scalability and performance (Drake, 2019). Examples for key-value store databases are Redis, Memcached and Riak.

#### 2.4.2.3 Column-Oriented Database

Column-oriented databases or columnar databases are databases that store data in terms of columns. Unlike SQL databases, it groups data in terms of columns rather than tables. It stores the data in the column in record order, which means the index of a specific record in one column is the same for the other column. This allows queries to only retrieve data from selected columns rather than reading all rows and remove the unselected column of data. However, the drawback of this type of database is that it requires more time to insert a new record as separate write operations are required for all columns (Barnhill and David, 2021). Apache Cassandra is a popular example of NoSQL DBMS utilising this data storage model.

#### 2.4.2.4 Graph Database

Graph databases are considered a subcategory of document store databases. A graph database stores its data in documents and does not require a predefined schema for its data. However, graph databases add another abstraction layer to allow relationships between their documents. Additional concepts are introduced in graph databases to augment the capabilities of document store databases which include nodes, edge and properties. Nodes are independent records that can hold properties. A node is synonymic to the concept of a document in a document store database. The nodes relations are stored using edges. Edges represent how two nodes are related to each other (Vázquez, 2019). In some DBMS, edges can also have properties to store the unique characteristics of the relationship. Edges can be undirected or directed to indicate the direction of the relationship. Labels are also introduced in some graph databases to allow users to group nodes and edges by categories. Due to its capability to clean and simple ways of defining relations, graph databases are widely used for fraud detection, network routing, identity and access management (IAM) and full-text search applications. As of April 2021, Neo4j is the most popular graph modelled NoSQL database (Solid IT, 2021).

## 2.5 Database Synchronization

The process of synchronizing data to achieve consistency among different data sources is called data synchronization (Shodiq et al., 2015). Database synchronization is the synchronization of data in a database. There are two categories of data synchronization techniques which are unidirectional synchronization and bidirectional synchronization (Jindal, 2016). Unidirectional data synchronization copies and replace the data in the destination database from a source database. Bidirectional data synchronization merges the data from both source and destination databases. Conflict in the source and destination data will cause bidirectional data synchronization to fail.

## 2.6 BATMAN-ADV

BATMAN is a proactive routing protocol and system for a mesh network. BATMAN-ADV is a newer implementation of BATMAN. on layer 2. BATMAN uses an efficient algorithm that avoids redundant knowledge of the network and reduces overheads caused by network signalling (Pinto et al., 2010). BATMAN-ADV supplies user with a command-line interface (CLI) called batctl which helps users to set up a BATMAN-ADV network easily. BATMAN-

ADV nodes have three possible roles which are the routers, gateway and bridge. As BATMAN-ADV is a layer 2 protocol, it will not assign Internet Protocol (IP) address to the BATMAN-ADV interface, manual assignment or Dynamic Host Configuration Protocol (DHCP) server hosting must be done by the developer.

Routers in BATMAN-ADV is the basic node of the system. Any node in the mesh network is at least a router. Routers in BATMAN-ADV is responsible for re-transmitting or relay data packet to its destination within the mesh network. Routers will flood the network with originator messages (OMGs) to announce non-mesh clients, determine link quality and discover existing nodes (Abdalla et al., 2015). Applications can be hosted in the router itself.

When a router is configured into a gateway, it is capable to forward messages from the mesh network to other networks. This is commonly used to connect the network to the Internet. BATMAN-ADV gateway can include a DHCP server to provide an IP address for every newly joined mesh nodes dynamically.

For non-mesh devices to connect to BATMAN-ADV nodes, a bridge is needed. Bridges in BATMAN-ADV can be hosted on any interface of the router except for the interface used to connect to the BATMAN-ADV network. Bridges work as an AP for the non-mesh client to connect to the mesh nodes and also the networks connected to the mesh network via gateways. A DHCP server can be included to provide dynamic IP addresses to its associated non-mesh clients. An example of a BATMAN-ADV architecture is shown in Figure 2.10.

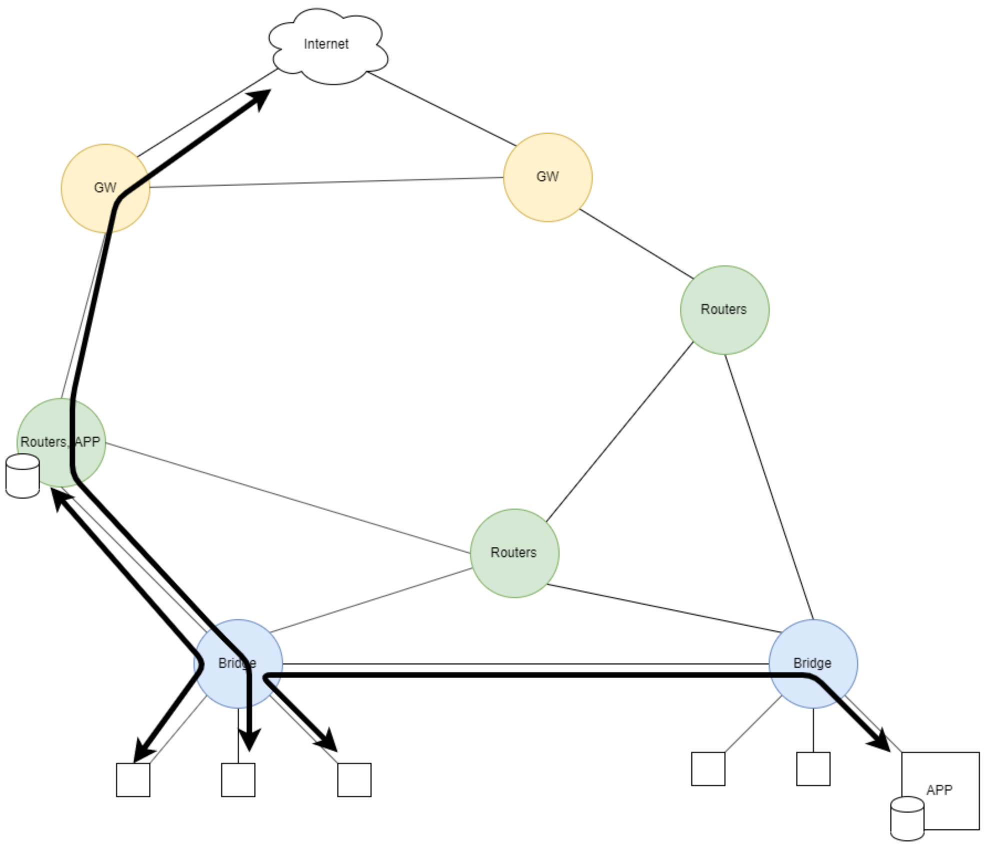


Figure 2.10: Example of a BATMAN-ADV Architecture.

### 2.6.1 Database Synchronization

There is no native database synchronization service in BATMAN-ADV. However, multiple database synchronization methods can be performed on the application layer.

## 2.7 NerveNet

NerveNet is first proposed to prepare a network infrastructure for the New Generation Network (NWGN) (Owada, Inoue and Ohnishi, 2011). Developed by Japan’s National Institute of Information and Communications Technology (NICT), NerveNet works as a regional information sharing platform and network which is disaster-resilient without depending on cloud applications on the Internet (Owada et al., 2019). NerveNet targets to migrate regional applications from the cloud to the local region and convert the current regional tree topology network to a mesh topology network to increase the resilience of the network (Inoue and Owada, 2017). The components of the NerveNet are network base station (BS), gateway (GW) and network manager (NM).

NerveNet nodes can connect to each other via wireless interfaces like Wi-Fi or LoRa or wired interfaces such as Ethernet or fibre optics.

NerveNet base station is the basic member of the mesh network which is responsible to work as a wireless AP. Non-mesh devices are able to connect to the NerveNet application by connecting to the nearest base station. Applications can be hosted in the NerveNet base station itself or hosting on its dedicated hardware and connect to a NerveNet base station.

During the initial setup of the network or when there is severe damage to the network, NerveNet's network manager will be responsible for the initial settings and handover logic of the whole network (Owada, Inoue and Ohnishi, 2011). To connect NerveNet to another network such as the Internet, NerveNet's gateway is used. Multiple gateways can be established in the network to avoid single-point failure. If there is at least one gateway node setup in the NerveNet, all the nodes and clients can connect to the other network. An example of a NerveNet architecture is shown in Figure 2.11.

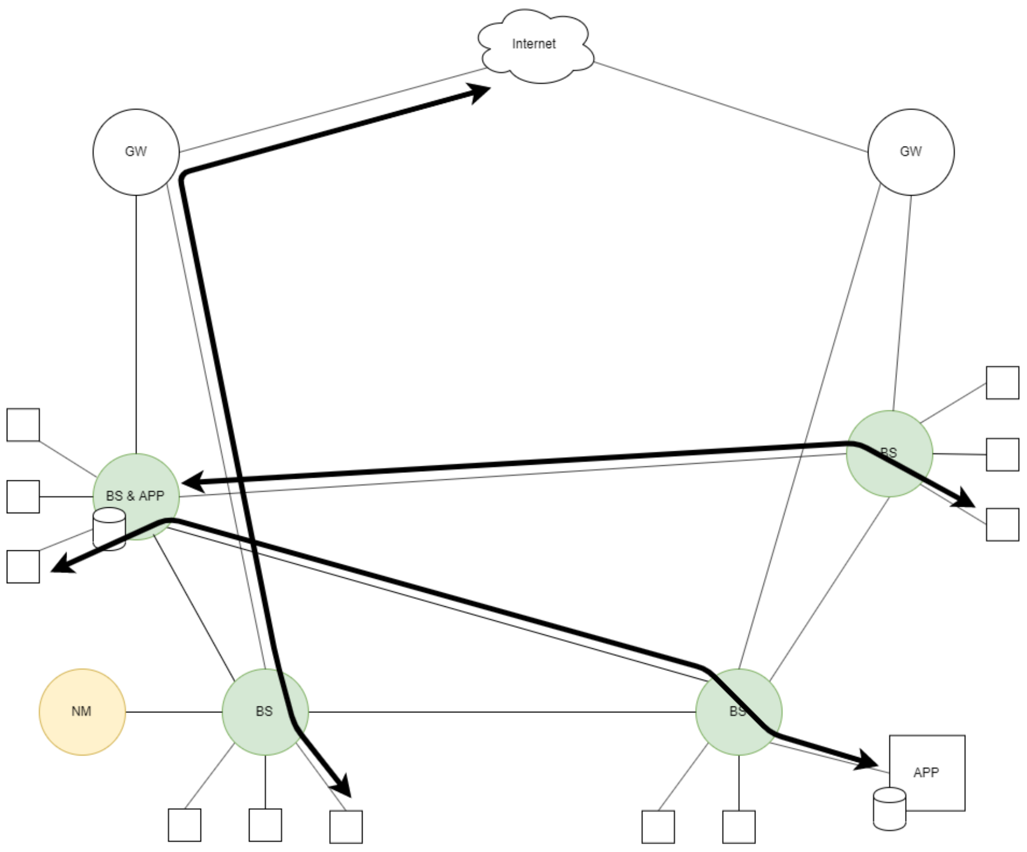


Figure 2.11: Example of a NerveNet Architecture.

### 2.7.1 Database Synchronization

Database synchronization in NerveNet is native and automatic for its base station. NerveNet comes with two databases that store routing information and application data, respectively. The database which stores application data can be used by application developers to synchronize its database throughout the whole NerveNet. The synchronization process is handled by the NerveNet Operating System (OS). Non-mesh application servers can also connect to the NerveNet by using an API developed to push data to the nearest base station's database.

NerveNet database is synchronized via the hearsay daemon which compares the hash of other databases with its node's database. Hearsay daemon synchronizes the MySQL database shipped within the NerveNet distribution. However, the Hearsay daemon only synchronizes Insert and Update actions in existing tables. To create a custom table, an Extensible Markup Language (XML) file must be created in the directory '/writable/etc/tables.d/'. The XML file defines the fields (or column) of the table as well as the naming of the table. To synchronize files such as images and documents, a special table is predefined in the MySQL database called 'shbt_boxshare'. This predefined table can be used to synchronize file by using the 'attached' field. At the receiving end, the file will have a different unique name and the path is shown in the synchronized database of the receiving node.

## 2.8 Communication Protocols

Communications in a computer system are often very complex and require high levels of encapsulations. To simplify the process of sending data from one process to another process or from one device to another device, the Open Systems Interconnections (OSI) model is created to standardize the concepts of different communication functions in telecommunication, computing system and machines. Communication protocols define a format and rules for two network entities to exchange digital information with each other. The OSI model generalizes and categorizes various communication protocol into 7 abstraction layers which include the application layer, presentation layer, session layer, transport layer, network layer, data link layer and last but not least, the physical layer which can be seen in Figure 2.12.

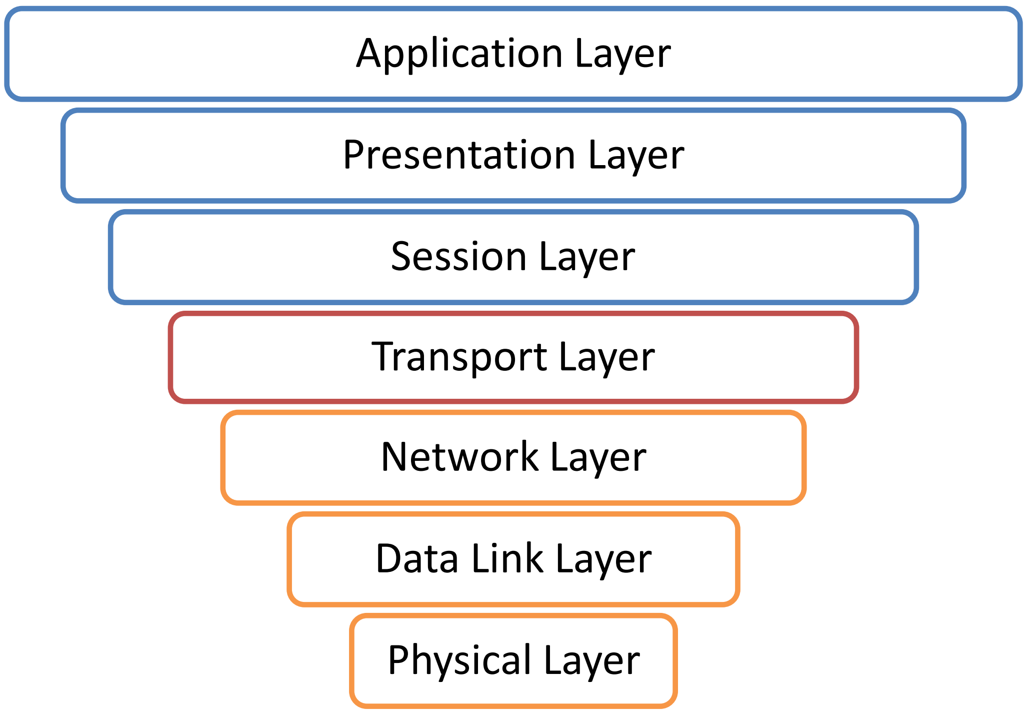


Figure 2.12: An Illustration of the OSI Abstraction Model.

Each layer in the OSI model serves different purposes in computer communication. All higher layer in the OSI model depends on its next lower layer to perform primitive functions. The OSI model can be further grouped into two separate groups divided by the transport layer. The software layer consists of all of the upper layers such as the application layer, presentation layer and session layer while the hardware layers consist of all of the lower layers including the network layer, data link layer and physical layer. The software layer protocols are implemented solely by software. These protocols mainly handle application issues such as preparing the data to transmit, track on application session and securing the data from malicious access and alteration of communicated data. On the other hand, the hardware layer protocols deal with problems related to the actual data transfer. Except for the network layer protocols, the hardware layer protocols are implemented using a combination of software and hardware standards. The transport layer is the centre of the OSI model. It provides functions that ensure a reliable link by providing packet segmentation, controlling the packet flow and handling network errors (Layers of OSI Model Explained, 2021). Common protocols for each OSI model layers are shown in Table 2.1.

Table 2.1: List of Common Protocols for Each OSI Model Layers

| OSI Layer | Protocols |
|---|---|
| Application | HTTP, MQTT, SMTP, FTP, DHCP |
| Presentation | TLS, SSL, SSH |
| Session | RPC, SDP, SOCKS, ZIP |
| Transport | TCP, UDP, DCCP |
| Network | ICMP, OSPF, AODV, ARP |
| Data Link | batman-adv, PPP, IEEE 802.11, Ethernet |
| Physical | Ethernet, USB, CAN bus, SPI, $I^2C$ |

## 2.9 Scalability

The scalability of a software or application is the capability of the system to maintain its performance at a steep increase in workload without refactoring the system. Scalability is a crucial factor to applications that handle multiple simultaneous users and concurrent requests and will determine the adoption rate of the application. When the system is not scalable enough, it will place huge stress on the current application architecture and plummet its performance. This directly affects the experience of the application users.

If scalability is considered at the beginning of the development stage, it is much easier and requires lesser resources to implement an efficient system. There are a few limiting factors that affect the scalability of an application which includes the maximum stored data, code quality and processing capabilities (Concepta, 2019). For applications that require storage of increasing size persistent data, the maximum data storage capacity of the application must be put into consideration. There are a few existing strategies to improve the scalability of databases, which include query optimization, vertical scaling, database replication, partitioning and database sharding (Nath, 2019).

Scalability must also be incorporated in the code architecture by ensuring simple integration of future features. The code of a scalable system must also stick to the “Don’t Repeat Yourself” (DRY) principle and avoid “spaghetti code” (Arya, 2017). To further improve the application process, asynchronous processing can allow your application to fully utilize its resources (Manandhar, 2020).

Processors and random-access memory (RAM) of the server or application host are also two potential bottlenecks when it comes to multiuser applications. There are two methods of scaling processing resources of applications which include vertical scaling (scale-up) and horizontal scaling (scale-out). In terms of vertical scaling, in event of resource deficiencies, additional or stronger hardware is used to increase the performance of the existing machine. On the other hand, horizontal scaling increases its capability by adding additional machines to the system and replicating its existing system to the newly added machines (Manandhar, 2020). A load balancer is then used in horizontal scaling to distribute the workload between all its machines.

## 2.10 Monolithic Architecture and Microservice Architecture

Application architecture involves the design pattern and structure of the application. Selecting the suitable application architecture is important for the scalability, performance and development efficiency of the system. However, there is an overwhelming amount of software architecture that can be used and sometimes a union of architecture is required to achieve the application requirements. Among all architectures, the two most discussed architectures are monolithic architecture and microservice architecture.

Monolithic architecture structures all its components in a single entity. Its code is structured into modules within one single code base. Deployment of monolithic applications is also very simple as it only requires executing a single set of command. The downside of monolithic architectures is that it can be very expensive to scale the system. When scaling monolithic applications, replication of the whole system is needed even if only one function requires scaling. Large monolithic programs also get very challenging to manage. As monolithic applications are tightly coupled, it can get hard to understand for new members of the team. It is also hard to modify tightly coupled components.

Microservice architecture segregates the application into smaller components, each handling one or more functions of the application. These smaller components are deployed in separate services and communicate with each other using a previously agreed set of protocol and messaging format. Scaling for microservice is easier compared to a monolithic application as each service can be individually deployed and scaled horizontally. Deployment of microservice application can be difficult due to their number of services. With the aid of Docker and Kubernetes, microservice deployment is getting much simpler. It is also much easier to understand and modify microservice applications as each service is rather small. Another benefit of microservice architecture is the capability for multiple development teams to work on different parts of the application concurrently.

## 2.11 Server and Cloud

In a client-server model distributed architecture, the application distributes its workload to individual clients or users to reduce the stress of a central service called a server. The individual client shares the resources and services provided by the server. A server can be a single machine or a cluster of machines that run a server program. A server program is software that listens to requests from a set of clients using protocols such as HTTP, FTP and SMTP. An illustration of a client-server model software architecture is shown in Figure 2.13.

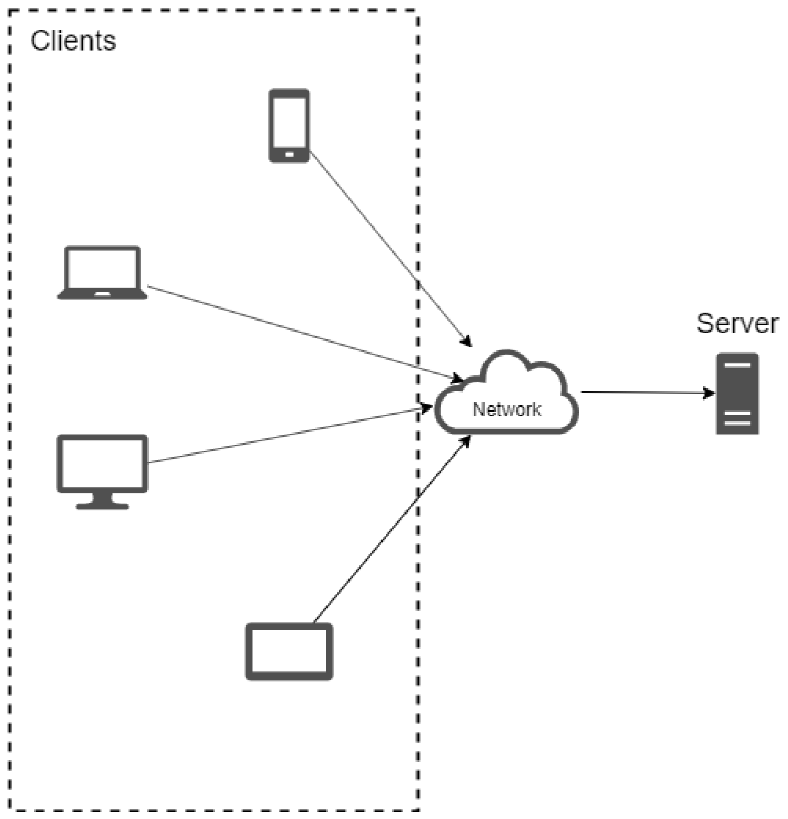


Figure 2.13: An Illustration of Client-Server Model Application Architecture.

In a client-server model, codes that are executed on the client-side is called the frontend code base while codes that are executed on the server-side is said to be the backend code base. Data are usually stored in the server to allow cross-device access. Data are also pre-processed on the server-side before sending it to the client.

Cloud is a server that managed by companies in data centres and is rented to other companies or individuals. Traditional servers are deployed in dedicated servers which require constant maintenance and management. With cloud services, software companies and individual developers do not need to manage the hardware maintenance. Common cloud service providers and platforms are Digital Ocean, Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP). The user accesses and deploys their services into the rented server through the Internet via protocols such as Secure Shell (SSH) Protocol, Secure Copy Protocol (SCP) and Secure File Transfer Protocol (SFTP).

## 2.12 Cloud Computing and Edge Computing

As discussed in the previous section, cloud computing is a rented server hosted by cloud service providers on the Internet. This allows fast deployment of servers on the Internet which allow worldwide use of its services. However, there are some delay-sensitive cases where cloud computing is not suitable. To meet such requirements, the service must be deployed in the client itself or an in-premise server near the source of the data. Such computing distribution is called edge computing. Other use cases that require edge computing is when the service must still be available during a network failure to the Internet such as disaster response applications, evacuation systems and security systems (Arora, 2021). A mix of cloud services and services in the edge is called a hybrid cloud strategy.

## 2.13 Digital Twin

A digital twin is a virtual copy or model of a process, service or device (Marr, 2017). Digital twining maps the interfaces of the physical entity such as commands, properties, components, telemetries and relationships of the physical entity to a digital version, bridging the gap between the physical and digital world. With a digital twin, cloud services can simply interface with the physical device using the digital twin services. Some of these services are provided by cloud service providers such as AWS Device Shadow service, Microsoft Azure Digital Twin and digital twin service from GCP IoT Core. There are also open-source versions of such services like Eclipse Ditto. Some of these services use descriptive language to define the mapping of the interfaces such as Digital Twins Definition Language (DTDL) for Azure and Vorto language for Eclipse Ditto. These descriptive languages can be used by smart sensor manufacturers, machine maker and system integrator to define the digital twin interfaces of their products. These predefined configurations can then be used by software developers to integrate cloud services into the existing system and allow simple integration of smart services into existing systems. Thus, digital twins are very important in IoT, smart cities and smart manufacturing developments.

## 2.14 User Interface

The user interface is a set of components that allow human to interact with machines and software. It includes the hardware where the human can interact with such as touch screens, monitors, keyboards and software. The software user interfaces can be developed using various technologies depending on the OS and the hardware of the devices. Three common user interface categories in computer software development include native application, hybrid applications and web applications. A native application is an application that is built specifically for a certain OS. This includes developing android applications in Java and building iOS applications using Objective-C or Swift. Web applications are developed using web technologies such as Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (JS). It requires clients to have a web browser to view its interface. However, due to the popularity of web technologies, most of the devices have certain web browsers which makes it good for situations that require cross-platform integrations. For hybrid applications, it is developed using web technologies and is packaged in a native wrapper (YML, 2020). Examples of hybrid application frameworks are Ionic, Flutter and React Native. Some hybrid frameworks like Flutter can even access native elements such as the location and camera function of devices. Hybrid application frameworks are very powerful as it allows developers to reuse most of the code for all platforms.

## 2.15 Summary

Disaster tolerant networks need to be redundant and resilient to faults. To accomplish that, mesh topology can provide resilience to the network. Redundancy can be provided by data synchronization and backup in all nodes or selected nodes in the mesh network. Two mesh network options are listed which include NerveNet and BATMAN-ADV. The advantage of NerveNet over BATMAN-ADV is that it has native support of data synchronization. For a regional disaster response network, a cloud monitoring dashboard is needed to allow nationwide monitoring and control. For a reliable nationwide monitoring cloud, the design of the server-side architecture must consider the scalability and performance of the server while remaining relatively simple. All these considerations lead to the design done in this project.

# CHAPTER 3

# METHODOLOGY AND WORK PLAN

## 3.1 Introduction

A hypothetical regional disaster-resilient network for disaster data propagation and backup is developed using NerveNet. Disaster sensor data (which includes camera images, victim count and disaster detected) is synchronized throughout the network and also published to a cloud monitoring server. The purpose of having a cloud monitoring service is to prepare for larger catastrophic events that span more than just a region. The capability of each single-hop link will be analysed.

## 3.2 Work Plan

The part 1 of the final year project consists of four stages, namely literature review and related knowledge studies, system architecture design and model testing using BATMAN-ADV with client-server MongoDB database replication, deployment of MongoDB database replication in BATMAN-ADV mesh network and deployment of NICT's NerveNet network. In the initial stage of the project, a literature review on networking has been done. At the stage of system architectural design and model testing using BATMAN-ADV with client-server MongoDB database synchronization database replication method, a database replication system in the BATMAN-ADV network is developed. Deployment of MongoDB database replication in BATMAN-ADV mesh network will then take place to test the database replication of MongoDB. Lastly, NICT's NerveNet is to be deployed on a small scale for knowledge verification purposes. However, the deployment of NICT's NerveNet cannot be done on time and have to be delayed to the second part of this final year project. This is due to the delay from Japan's NICT on providing the required resources and support.

Similarly, there are four stages in the second and final part of the final year project which include NerveNet testbed setup, development of the cloud dashboard's digital twin service, development of the cloud dashboard data API and frontend, and the NerveNet testbed's data synchronization feature. At the first stage, guidelines and documentations for a simple setup of a three-node NerveNet testbed are provided from Japan's NICT. The documents are studied and applied to set up a three-node NerveNet mesh network. At the second stage, a simple device twin service is created for the cloud server to collect node statuses, which will be stored in a database for monitoring use. The monitoring server API and frontend are then created to provide a human-readable dashboard. While developing the API and web application, the data synchronization feature of NerveNet is studied and deployed concurrently. The flow chart of the work plan for the final year project is shown in Figure 3.1.

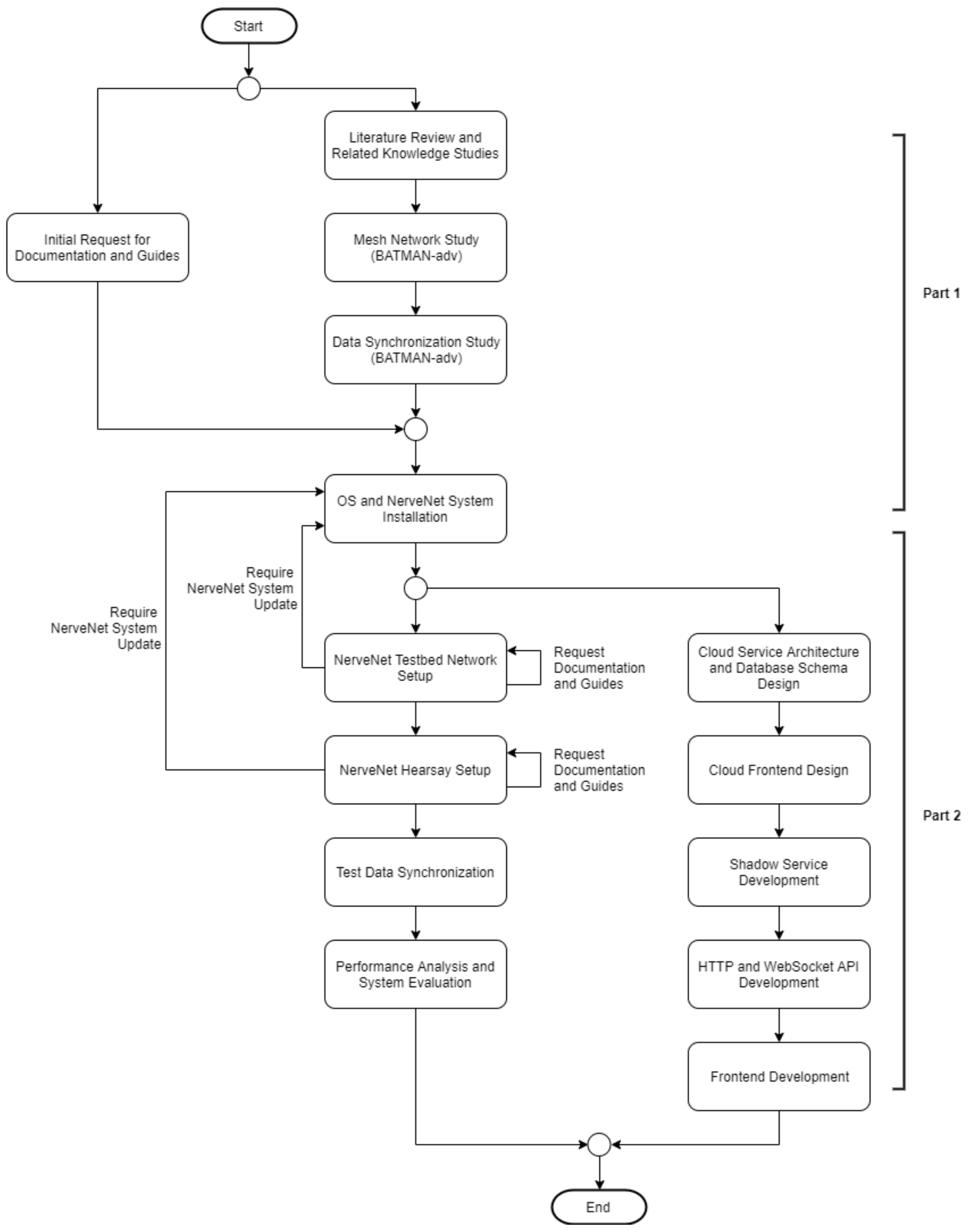

Start
Literature Review and Related Knowledge Studies
Initial Request for Documentation and Guides
Mesh Network Study (BATMAN-adv)
Data Synchronization Study (BATMAN-adv)
Part 1
OS and NerveNet System Installation
Require NerveNet System Update
Require NerveNet System Update
NerveNet Testbed Network Setup
Request Documentation and Guides
Cloud Service Architecture and Database Schema Design
NerveNet Hearsay Setup
Request Documentation and Guides
Cloud Frontend Design
Part 2
Test Data Synchronization
Shadow Service Development
Performance Analysis and System Evaluation
HTTP and WebSocket API Development
Frontend Development
End

Figure 3.1: Flow Chart of the Work Plan for the Final Year Project.

## 3.3 Mesh Network

NICT’s NerveNet is selected for the use case of the disaster response network. NerveNet is advantageous for this use case as it provides a database synchronization feature off the bat. For preliminary studies on mesh network with database synchronization, BATMAN-ADV is chosen due to its popularity and setup simplicity.

## 3.4 NerveNet Testbed Application

The testbed is modelled to be used as a regional disaster response and monitoring platform. This platform includes applications for disaster response, sensor nodes, and broadcast devices. The testbed is developed using NerveNet as the OS of its mesh infrastructure. An illustration of the NerveNet based disaster response and monitoring platform is shown in Figure 3.2.

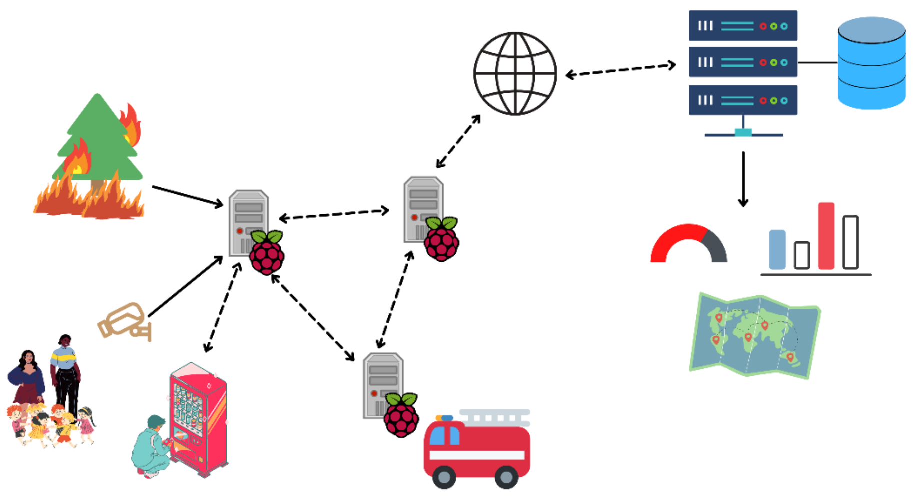


Figure 3.2: An Illustration of the NerveNet based Disaster Response and Monitoring Platform.

When a sensor node detects a threat like high ambient temperature or fire in a forest area, it will store its telemetric data in two locations, the cloud monitoring server and its local database. Its telemetric data will then be synchronized to the other devices that are within the mesh network. This ensures the resiliency of the collected data even if the sensor node itself is destroyed. If all gateway nodes of the mesh network malfunction, there are still copies of the data in member nodes within the mesh network. A node can have multiple data schemas. Member nodes are nodes that have the same data schema.

This synchronized data can then be used by relevant emergency response agencies to evaluate the threat level and damages as well as to prepare a response to the problem. Authorities can then trigger evacuation broadcast and notice to certain area devices such as vending machines, billboards and evacuation alarm depending on the threat level. At low threat levels, quick responses can even mitigate the threat before it becomes a disaster.

During a catastrophic disaster, there is a possibility that the whole regional network will be interrupted. At such an event, the federal authorities must be informed to provide support to the disaster zone. The latest state of the disaster zone must be recorded to allocate sufficient rescue resources to the location. It is provided by the cloud monitoring server in this disaster response and monitoring framework.

## 3.5 Proof of Concept

For a better understanding of NerveNet, BATMAN-ADV has been used to study the concept of database synchronization in a mesh network. Furthermore, it is a layer 2 mesh network that is similar to NerveNet.

### 3.5.1 Network Design

A simple BATMAN-ADV network is set up using three raspberry pi 3, model B+. The raspberry pis run Raspbian Buster Lite OS and are configured to work as a gateway, a basic router with an application server and a bridge respectively. The gateway is connected to a local network through a switch and is accessible through a 192.168.0.0/24 network which is also connected to the Internet via a router. The BATMAN-ADV network is configured to operate with Wi-Fi ad-hoc mode on the 11$^{th}$ channel of the 2.4GHz wireless band. The channel is selected to reduce signal interference from another wireless network. The BATMAN-ADV mesh network is configured to be on a 192.168.10.0/24 network.

As there is only one wireless interface in-built into the raspberry pi, the bridge node is configured to have BATMAN-ADV to operate on Wi-Fi while the ethernet port in bridge to the network to allow non-mesh clients to connect to the mesh network. However, this limits the bridge node to be able to connect to only one non-mesh client which is not ideal. This problem can be solved by either using a USB wireless interface extension or using a wireless AP device. In this case, the wireless AP device is selected as it is the only option available by that time. The wireless AP is connected to the bridge's ethernet port and is configured to provide a DHCP server, and it is on a 192.168.0.0/24 network. This may seem like a network overlap with the local network, but it is not the case as BATMAN-ADV gateways will only allow outgoing connections from the mesh network or its non-mesh nodes and blocks all incoming connections.

Web applications can be hosted on the router which is neither a gateway nor a bridge. This node application can be accessed within the mesh network including the non-mesh nodes by referring to its IP address or MAC address. The deployment architecture is shown in Figure 3.3.

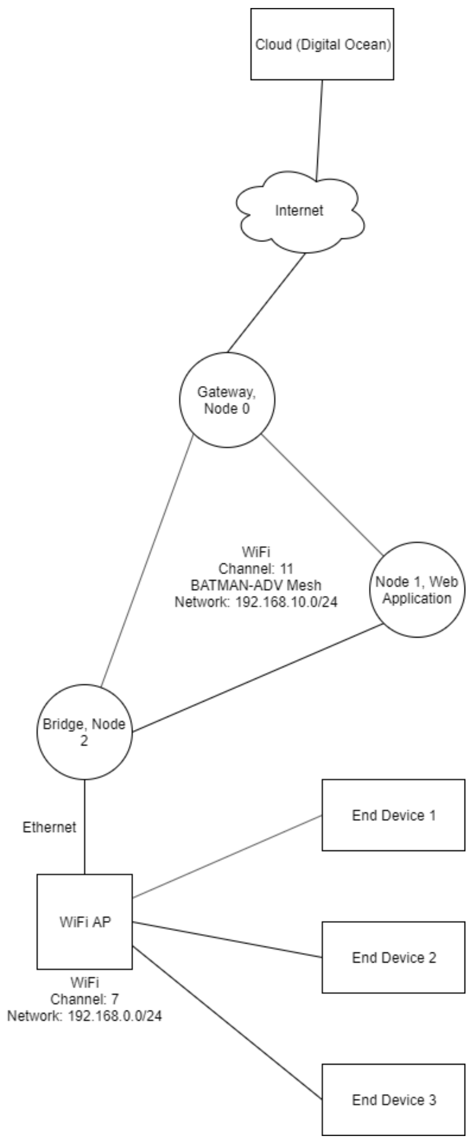


Figure 3.3: Simple BATMAN-ADV Deployment.

### 3.5.2 Database Synchronization Methods

There are four proposed architecture and systems for database synchronization using BATMAN-ADV. However, only one of the four proposed methods is developed and evaluated. The four architecture are listed below.

(i) Client-Server Model Database Synchronization
(ii) Source Broker Publish-Subscribe Model Database Synchronization
(iii) Receiver Broker Publish-Subscribe Model Database Synchronization
(iv) Database Replication Provided by the DBMS

The client-server model database synchronization method is created by having a Hypertext Transfer Protocol (HTTP) client at the data source and having HTTP servers in all receiving nodes. The data source will have an event listener to MongoDB’s oplog. This event listener will queue any data changes in the database to a queue. The data in the queue will then be sent to all receiving nodes by using an updater. At the receiving end, a receiver will accept the update request and its relevant fields. The receiver will then perform the changes to its local database. The system architecture is shown in Figure 3.4.

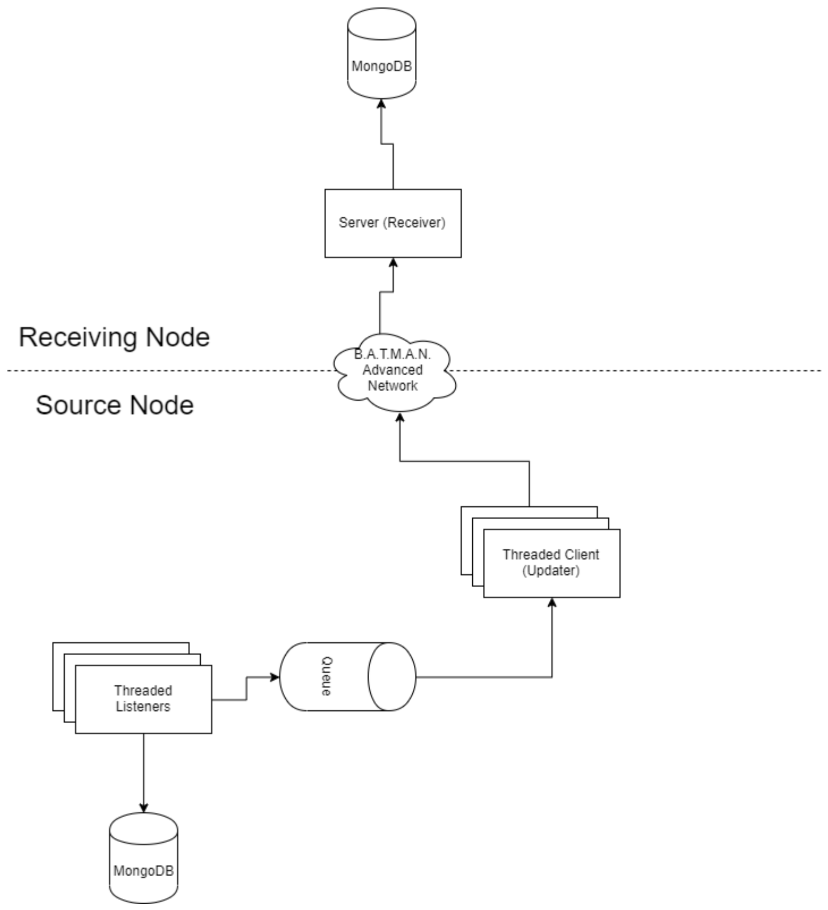


Figure 3.4: Client-Server Model Database Synchronization System Architecture.

Message Queuing Telemetry Transport (MQTT) protocol is used in the design of the publish-subscribe model database synchronization system. When there is an update in the source database, threaded listeners will update the MongoDB oplog to a queue. The publisher will then publish the data to a topic of the MQTT broker. If any nodes subscribed to the same topic, it can then receive the database changes and update the database respectively. The MQTT broker can be placed in the source node to form a source broker publish-subscribe model database synchronization system or placed in the receiving nodes to form a receiver broker publish-subscribe model database synchronization system. The system architecture is shown in Figure 3.5 and Figure 3.6 for source broker publish-subscribe model database synchronization and receiver broker publish-subscribe model database synchronization respectively.

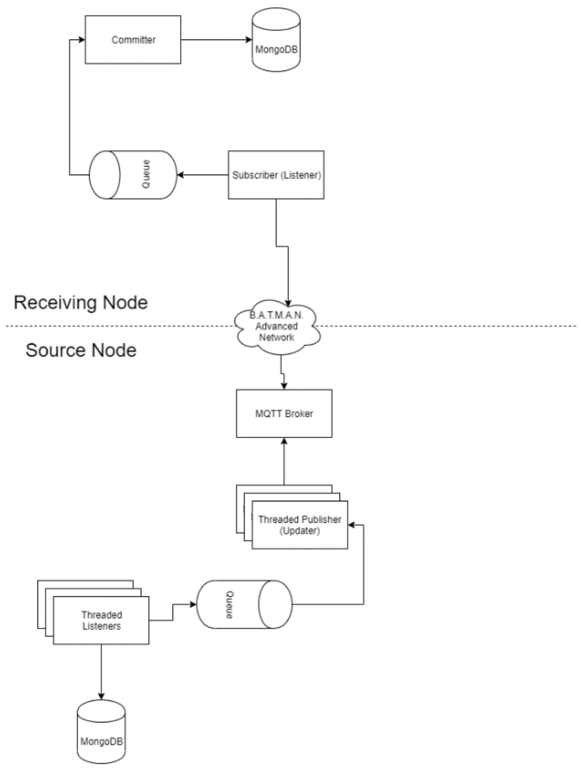


Figure 3.5: Source Broker Publish-Subscribe Model Database Synchronization.

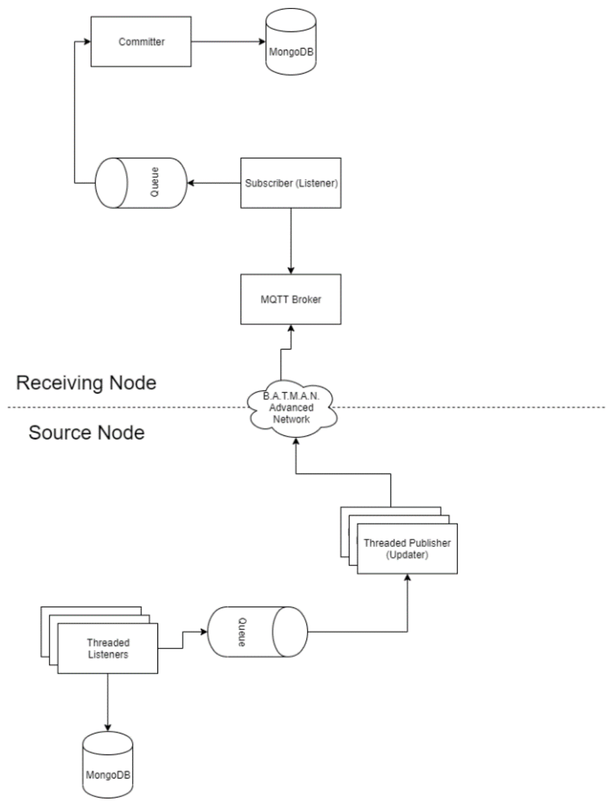


Figure 3.6: Receiver Broker Publish-Subscribe Model Database Synchronization.

By considering the complexity of all methods, the client-server model database synchronization method is selected among the four methods.

## 3.6 NerveNet

NerveNet is used as the mesh network for the testbed. NerveNet is a closed source, powerful mesh network shipped with plenty of useful features such as database synchronization. This makes it the best candidate for disaster resilient applications as we do not want application engineers to bother implementing low-level features and focus on the application itself. NerveNet is relatively new compared to other mesh networks and is still in active development by Japan's NICT.

### 3.6.1 Hardware Selection

The hardware used for deploying a three-node NerveNet testbed are listed below.

(i) 3 units of Raspberry Pi 3 Model B+

(ii) 3 units of TP-Link AC1300 Archer T4U High Gain Wireless MU-MIMO USB Adapter
(iii) 3 units of microSD card
(iv) 1 unit of Cat.5e Ethernet cable

As NerveNet is relatively new, the deployment of NerveNet can only be done using the OS provided by NICT which includes the features of NerveNet by the time of this project. Fortunately, a Raspbian version of the OS is provided. Thus, three units of Raspberry Pi 3 Model B+ are used as the processing unit for the NerveNet nodes as well as the sensor node. It is supported by NerveNet and is well suited for our use case due to its relatively small size. Three additional USB wireless adapters are needed to establish the wireless link as the internal wireless interface have been used as the wireless AP of the node. One microSD card is required for each node as Raspberry Pi does not have a persistent memory inbuilt. A cat.5e ethernet cable is needed for the gateway node to have internet connectivity.

### 3.6.2 Wireless Mesh Network

There are two methods to establish a wireless mesh network using NerveNet. It can either be set up using 4 address mode or Ethernet Remote Bridge (ERB) of NerveNet. In this testbed, the ERB method is adopted as the external wireless adapter does not support the 4 address mode.

### 3.6.3 Testbed Network Architecture

A simple three-node NerveNet testbed is deployed. The network architecture of the NerveNet testbed is shown in Figure 3.7.

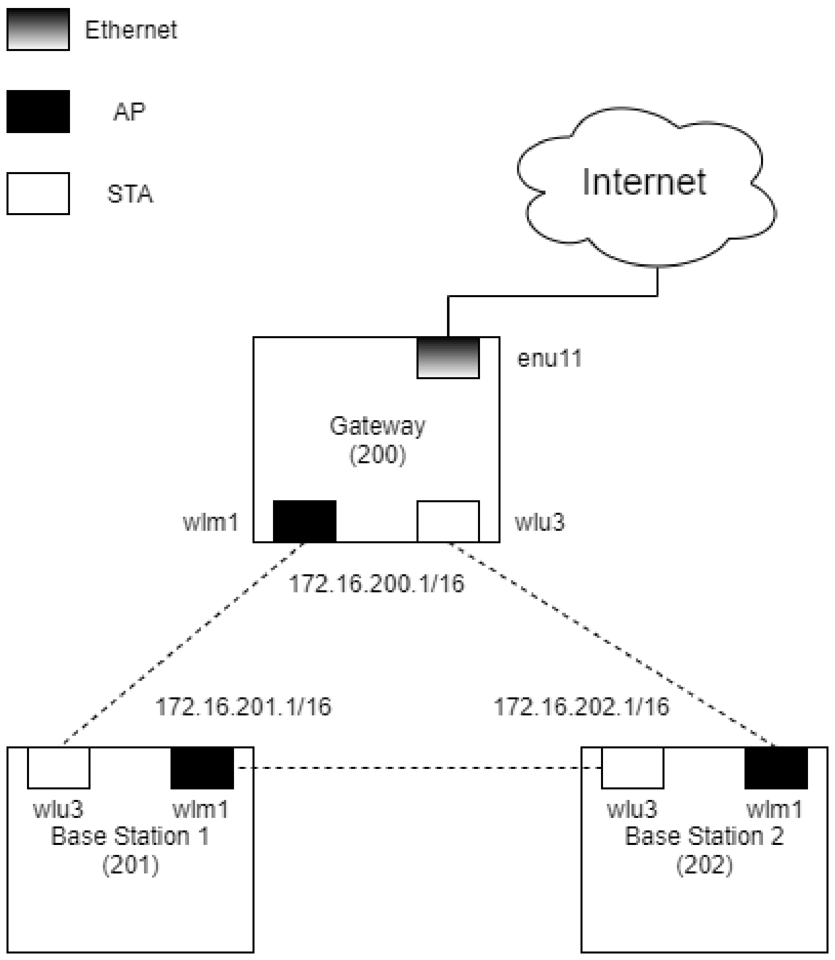


Figure 3.7: Network Architecture Diagram of the NerveNet Testbed.

The NerveNet testbed is designed to have one gateway node and two base station nodes. The wireless links of the NerveNet (indicated as a dotted line in Figure 3.7) are established using the ERB feature of NerveNet. To establish an ERB link between two nodes, one node must have a wireless interface configured as a wireless AP while the other node must have a wireless interface configured as its client. In ERB, unlike network routers, its wireless AP can only accept a connection from one client. Each ERB links are static and is defined with a collection of configuration files included in the NerveNet distribution. Based on Figure 3.7, the internal wireless interface of Raspberry Pi (labelled as “wlm1”) is configured as a wireless AP while the other external interface (labelled as “wlu3”) is configured as a client station (STA) of the adjacent node’s wireless AP. The gateway is similar to a base station with some additional configurations to enable re-routing packets to external networks. The gateway is configured to route external packets to the network at its Ethernet interface labelled as "enu11" in Figure 3.7.

In default, NerveNet is configured to work in the 172.16.0.0/16 network. The IP address of each node is 172.16.n.1 where n is the node id of the node defined during the installation of the network. For example, the gateway node in Figure 3.7 has a node id of 200 (written in parenthesis), thus, it has a NerveNet IP of 172.16.200.1.

## 3.7 Database Synchronization

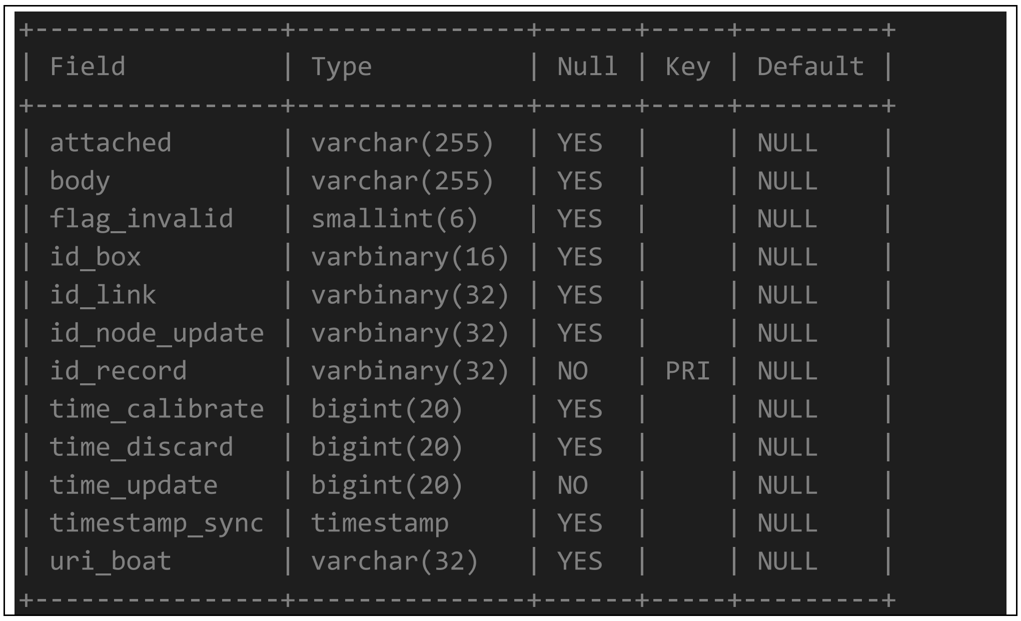
In NerveNet, a MySQL database is included. This database is mainly used as the data storage for synchronized data, but it can also be used for non-synchronized data. NerveNet Hearsay daemon synchronizes the data within the MySQL database based on the checksum of tables defined in the '/writable/etc/tables.d/' directory. When a table schema is defined in that directory using an XML template, Hearsay will automatically create the defined table into the internal MySQL database when initialized. Tables that are not defined in the '/writable/etc/tables.d' directory would not be synchronized. All member nodes of the table must also contain the same XML definition in its own '/writable/etc/tables.d' directory. The table definition for the disaster sensor application of the testbed is shown in Table 3.1.

Table 3.1: Disaster Sensor Application Hearsay Table Schema.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<database>
    <table id="0x0060" name="application_disaster">
        <dummy id="0xff02" label="record_id"/>

        <column id="0x0601" label="disaster_detected"
        name="disaster_detected"
        type="char"
        dbtype="VARCHAR(32)" />

        <column id="0x0602" label="victim_count"
        name="victim_count"
        type="int" size="8"
        dbtype="BIGINT NOT NULL"/>

    </table>
</database>
```

For image synchronization, a table, ‘shbt_boxshare’ is predefined by NerveNet. The table fields and description are shown in Table 3.2.

Table 3.2: The Table Fields of ‘shbt_boxshare’.



```
+----------------+--------------+------+-----+---------+
| Field          | Type         | Null | Key | Default |
+----------------+--------------+------+-----+---------+
| attached       | varchar(255) | YES  |     | NULL    |
| body           | varchar(255) | YES  |     | NULL    |
| flag_invalid   | smallint(6)  | YES  |     | NULL    |
| id_box         | varbinary(16)| YES  |     | NULL    |
| id_link        | varbinary(32)| YES  |     | NULL    |
| id_node_update | varbinary(32)| YES  |     | NULL    |
| id_record      | varbinary(32)| NO   | PRI | NULL    |
| time_calibrate | bigint(20)   | YES  |     | NULL    |
| time_discard   | bigint(20)   | YES  |     | NULL    |
| time_update    | bigint(20)   | NO   |     | NULL    |
| timestamp_sync | timestamp    | YES  |     | NULL    |
| uri_boat       | varchar(32)  | YES  |     | NULL    |
+----------------+--------------+------+-----+---------+
```

To synchronize any file (including images), the full path of the file is used as the field value of the field ‘attached’ in the new record. When committed, the file will then be synchronized to its member nodes and ‘shbt_boxshare’ table if the member nodes are updated with the corresponding local file path in its attached field. The file name and path will be different from its source. All synchronized file will be stored in the directory ‘/dev/shm/fieldfile/’.

## 3.8 Cloud Dashboard

A cloud-based web application is developed to provide sensor data and network statuses of multiple NerveNets. The development of the cloud dashboard is separated into two major parts which are the backend and frontend of the cloud dashboard. The backend development work includes the development of a simple digital twin system and a backend API for the consumption of the frontend dashboard while the frontend development focuses on the visual elements where users interact with.

### 3.8.1 Backend Development

The services required for the cloud dashboard are listed below.

(i) Digital twin for JSON encodable data
(ii) Digital twin for files such as images
(iii) A Database for persistent data storage
(iv) A Representational State Transfer (REST) API for consumption of simple data
(v) A Websocket API for file streaming
(vi) A static file server
(vii) An optional load balancer

A proposed design for the service architecture of the cloud monitoring application is shown in Figure 3.8.

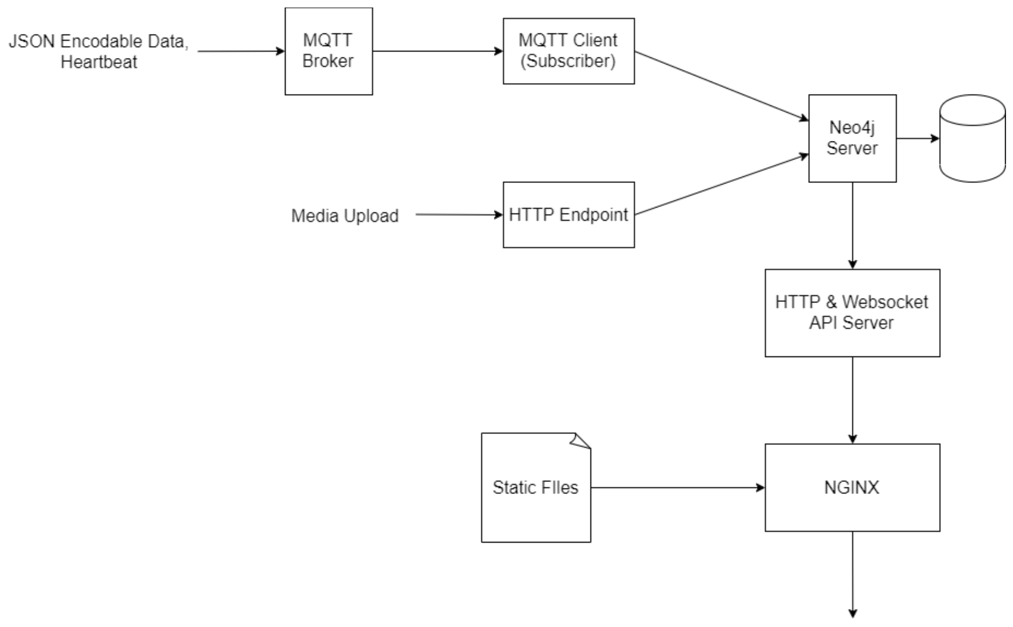


Figure 3.8: Service Architecture of the Cloud Monitoring Application.

The digital twin function is handled by two separate services which include an MQTT client for JSON encodable data and an HTTP server for media files. Image files can be either base64 encoded and sent via JSON format or sent directly in their binary form via HTTP. However, it is not recommended to encode media files into its base64 format as it can be very large. Thus, a separate endpoint that uses HTTP is created to receive media files. The media file history count can be defined in the server configuration file which will limit the number of stored media files to save server space. When the media storage limit has been exceeded, the device twin service will replace the oldest stored media data with the latest media data. The MQTT client listens to specific topics and format of which can be configured in the server. The topic format depends on the master template defined in the configuration file of the digital twin service and the topic field ‘topic’ of the Node record. All Node, Network and Sensor record are configurable via a device management CLI developed in Golang. The database used is a graph-based NoSQL database called Neo4j. The data and media records will be recorded in the database while the media file itself will be stored in the server memory. The data can then be retrieved via a RESTful API or a WebSocket API. The RESTful API is used for the retrieval of simple text data while the WebSocket API is used for file streaming. As the application is a web application, the Nginx static file hosting feature is used to serve the frontend static files. The Nginx server can also be configured to provide load balancing for all HTTP endpoints if needed. The technology stack of all backend services and tools are listed in Table 3.3.

Table 3.3: Technology Stack of All Backend Services and Tools in the Cloud Monitoring Server.

| **Backend Service** | **Programming Languages** | **Framework and Tools** |
|---|---|---|
| Digital Twin (HTTP) | Golang | GoFiber, Neo4j |
| Digital Twin (MQTT) | Golang | Mosquitto MQTT Broker, Neo4j |
| API Server | Golang | GoFiber, Neo4j |
| Static File Server | - | Nginx |
| Load Balancer | - | Nginx |
| Device Management CLI | Golang | - |

### 3.8.2 Frontend

The design phase of the frontend development is separated into three phases, namely sitemap designing, wireframing and visual designing. At the first sitemap designing phase, the application pages are listed out and connected using a simple block diagram. At the wireframing stage, a skeletal structure (wireframe) of the application is planned and sketched. The application design is finally styled based on the wireframe prepared in the visual design stage which can then be implemented using code. The sitemap, wireframe and visual design of the cloud monitoring dashboard application are shown in Figures 3.9, 3.10 and 3.11 respectively.

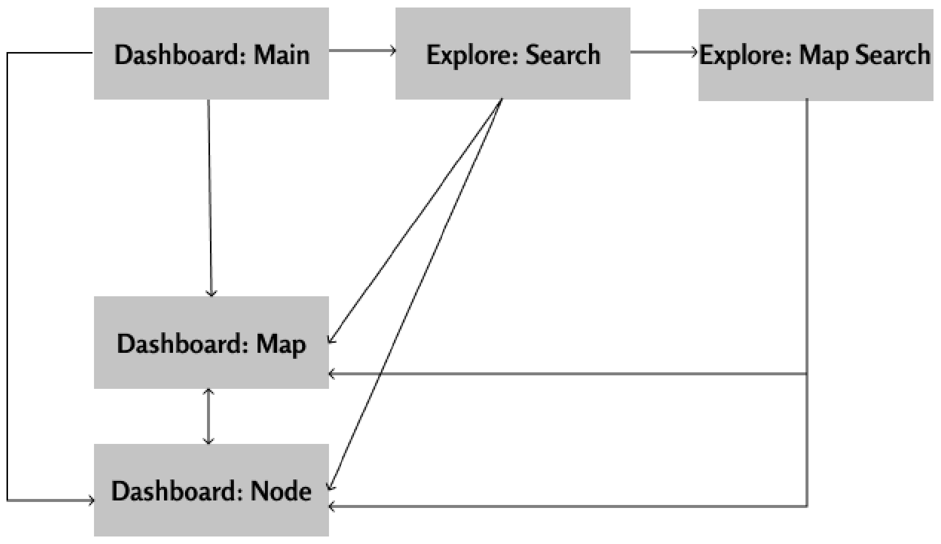


Figure 3.9: Sitemap of the Cloud Monitoring Dashboard.

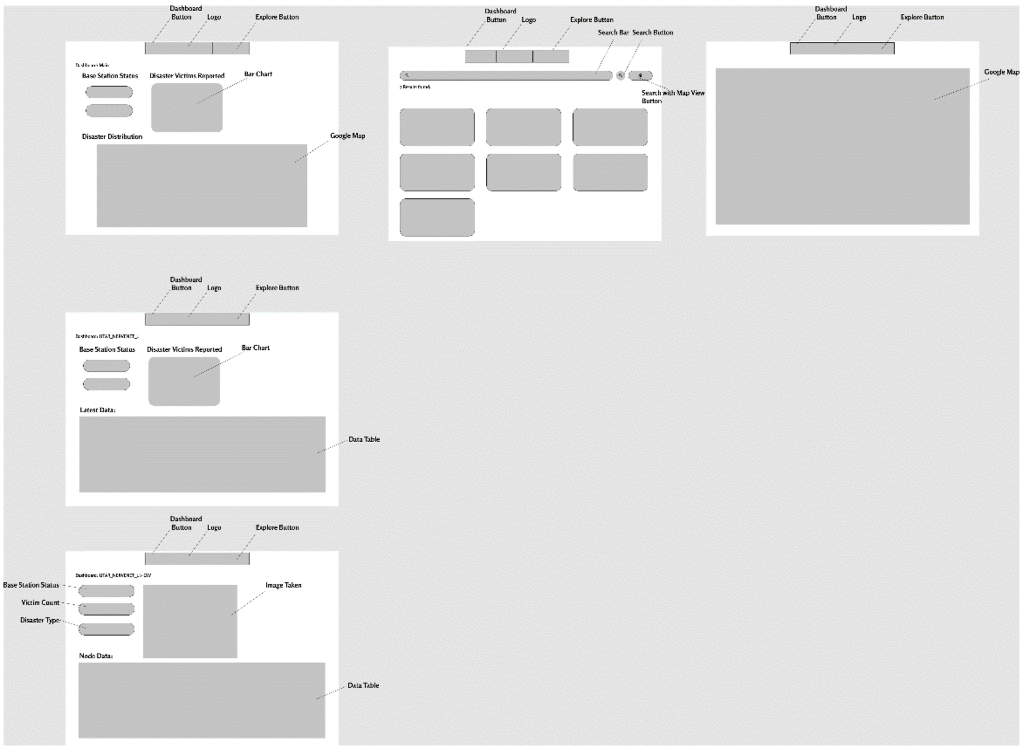


Figure 3.10: Wireframe of the Cloud Monitoring Dashboard.

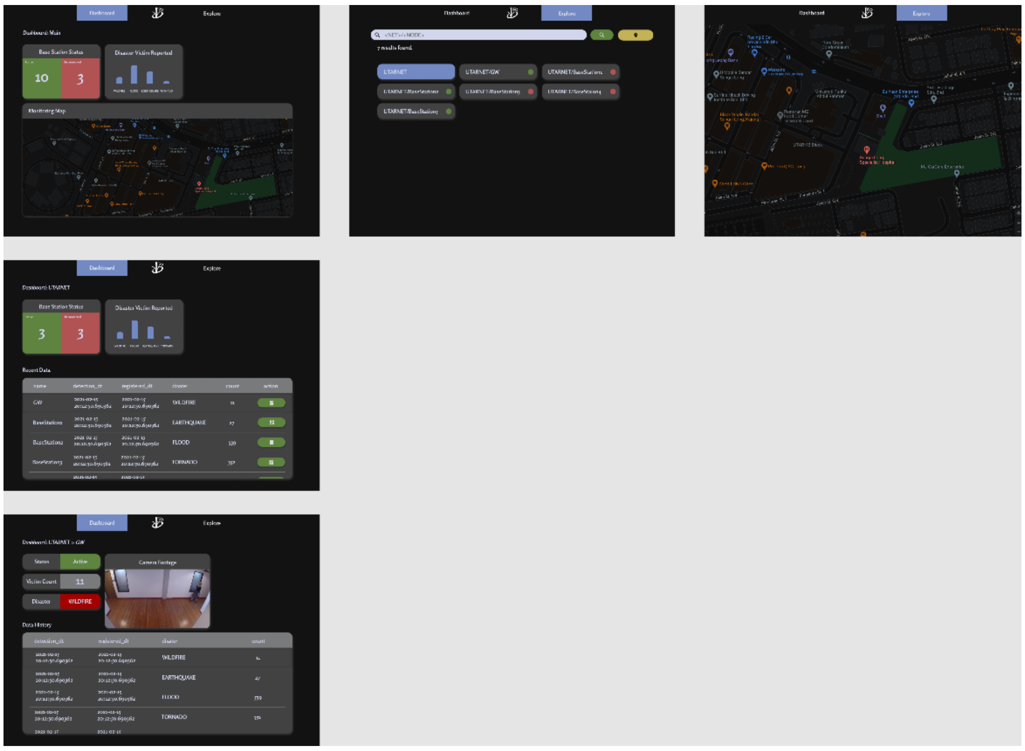


Figure 3.11: Visual Design of the Cloud Monitoring Dashboard.

## 3.9 Performance Test

An important criterion for efficient disaster response and monitoring platform includes a sufficiently performant network and framework. The platform consists of three major parts which is the NerveNet network, the Hearsay database synchronization function and the cloud monitoring dashboard. The cloud monitoring dashboard performance depends on the cloud server location, server specifications, and the NerveNet network performance. Thus, the performance of the cloud monitoring dashboard does not need testing.

### 3.9.1 Network Metric

There are two categories of network metrics, router metrics and performance metrics. Router metrics are used by routing devices to make routing decisions. Performance metrics, on the other hand, is used by network administrators, network managers and network architects to evaluate the network condition and performances. Some metrics can be utilized as both router and performance metrics. The common network metrics include packet delivery ratio (PDR), throughput, latency and jitter.

#### 3.9.1.1 Packet Delivery Ratio (PDR)

Packet delivery ratio (PDR) is defined as the ratio between the number of delivered packets and the total number of packets sent (Khan and Ramesh, 2019). A packet is considered delivered if and only if the packet is received in the receiving end. PDR value ranges from 0 to 1. A network is considered better if it has a higher PDR. This metric can only be used to evaluate the performance of the network. The equation of PDR is shown in Equation 3.1.

$$PDR = \frac{number\ of\ delivered\ packet}{total\ number\ of\ packets\ sent} \tag{3.1}$$

#### 3.9.1.2 Throughput

Throughput is the measured transmission rate of successfully delivered data. Throughput is often confused with bandwidth, but they are two different concepts. Bandwidth is defined as the maximum transmission rate to send data to its destination. Bandwidth is calculated theoretically but in a practical situation, the transmission rate would not achieve its rated bandwidth (Cox, 2020). On the other hand, the throughput is the actual measured transmission rate of data. Throughput may fluctuate depending on the network condition and may plummet due to network congestion, electromagnetic interference and route selection. It can be used for evaluating network performance as well as route selection. The equation of throughput is shown in Equation 3.2.

$$Throughput = bytes(delivered\ packets) \times \frac{8}{Finish\ time - Start\ time} \quad (3.2)$$

#### 3.9.1.3 Latency

In networking, latency is the time taken for some data to travel to its destination. It is often measured using the total time taken to travel to the destination and back to the source, also known as the round-trip time (RTT). Latency affects the throughput of the network and may even cause network packets to drop more often. Network latency may be affected by the distance between the sending and receiving end (Čandrlić, 2015) as well as network traffics. It can be used to rate the network's quality of service as well as determining the routing path to use in a router.

#### 3.9.1.4 Jitter

Network packet jitter also called packet delay variation (PDV) is the expected absolute value of the one-way delay variation between each sample packets as shown in Equation 3.3 (Dahmouni, Girard and Sansò, 2012). Assuming that $T_j$ is the one-way delay of the jth packet received in the receiving node, the jitter, J is then,

$$J = E\left[|T_{j+1} - T_j|\right] \quad (3.3)$$

However, since measuring one-way delay requires time synchronization, a method to estimate the interarrival jitter using RTT is defined in RFC 3550 section 6.4.1 and appendix A.8 (Schulzrinne et al., 2003). Assuming that $T_i$ is the RTT of the ith packet, where $i = 2,3,4, \ldots$, the approximated jitter from the first packet to the ith packet, $J_i$ can be written as

$$J_i = J_{i-1} + \frac{1}{16}\left(|T_i - T_{i-1}| - J_{i-1}\right) \tag{3.4}$$

This alternative method is used in tools such as iperf which does not require time synchronization between nodes. A high jitter value means the network will have a huge spike of latency, which will cause real-time application such as voice over internet protocol (VoIP), live video streaming and video conferencing to have data distortions, low PDR and bad overall quality of service.

### 3.9.2 Network Performance Test

The performance of the NerveNet base network is evaluated by measuring the following performance metrics in section 3.9.1. The tools and protocols used for measuring the metrics are shown in Table 3.4.

Table 3.4: Performance Metrics for NerveNet Network Performance Evaluation and the Tools and Protocol Used for the Measurement.

| Performance Metrics | Tools Used and Protocol |
|---|---|
| Latency | ping (ICMP) |
| TCP Throughput | iperf3 (TCP) |
| TCP Retransmission Count | iperf3 (TCP) |
| UDP Throughput | iperf3 (UDP) |
| UDP Jitter | iperf3 (UDP) |
| Packet Delivery Ratio (PDR) | iperf3 (UDP) |

The performance metrics are measured manually. Three tests are executed in order which includes an iperf3 Transmission Control Protocol (TCP) test, an iperf3 User Datagram Protocol (UDP) test and an Internet Control Message Protocol (ICMP) ping test. The parameters of all tests and the metrics obtained for all tests are listed in Table 3.5. Two phases of tests are to be done which are the single-hop and two-hop configurations. Each phase consists of 6 independent configurations and 5 iterations of the three tests are to be done. The phases and their configurations are shown in Table 3.6. For two-hop configuration, one link of the mesh network can be interrupted by removing the external wireless adapter of the link.

Table 3.5: NerveNet Network Performance Test, its Fixed Parameters and Obtained Metrics.

| **Test** | **Parameters** | **Metrics Obtained** |
|---|---|---|
| iperf3 TCP | Processes = 1<br>Test Duration = 30 seconds | • TCP Throughput<br>• TCP Retransmission Count |
| iperf3 UDP | Processes = 1<br>Test Duration = 30 seconds | • UDP Throughput<br>• UDP Jitter<br>• Packet Delivery Ratio |
| ping | Payload Size = 56 bytes | • Network Latency |

Table 3.6: Test Phases and its Corresponding Configurations.

| **Phases** | **Configurations** |
|---|---|
| Single-Hop Configuration Phase | Sender: GW, Receiver: BS1, Deactivated Node: BS2 |
| | Sender: GW, Receiver: BS2, Deactivated Node: BS1 |
| | Sender: BS1, Receiver: GW, Deactivated Node: BS2 |
| | Sender: BS1, Receiver: BS2, Deactivated Node: GW |
| | Sender: BS2, Receiver: GW, Deactivated Node: BS1 |
| | Sender: BS2, Receiver: BS1, Deactivated Node: GW |
| Two-Hop Configuration Phase | Sender: BS1, Receiver: BS2, Intermediate Node: GW |
| | Sender: BS2, Receiver: BS1, Intermediate Node: GW |
| | Sender: GW, Receiver: BS2, Intermediate Node: BS1 |
| | Sender: BS2, Receiver: GW, Intermediate Node: BS1 |
| | Sender: GW, Receiver: BS1, Intermediate Node: BS1 |
| | Sender: BS1, Receiver: GW, Intermediate Node: BS2 |

### 3.9.3 Database Synchronization Test

The telemetric data of sensors in a disaster response platform consists of text data and file data. To ensure the feasibility of NerveNet as disaster response and monitoring platform, its database synchronization latency must be sufficiently low for both file synchronization and simple text data. However, as synchronizing text data only requires Hearsay to synchronize the table fields, it is comparably requiring way lesser data transfer and resources than synchronizing file data. Due to time limitations, a latency test for file data synchronization is only required. The file data synchronization latency test is done by measuring the average delay time required for four different sized images to synchronize from GW to BS1 and BS2. The record is inserted into the local database of GW in order of the image size and the test is done for three iterations. The subsequent record is only inserted when the data is ensured to be successfully synchronized in BS1 and BS2. These tests are done manually and the average delay time of all data for each base station is recorded.

## 3.10 Summary

The mesh network used for the disaster response and monitoring platform is NerveNet. A proof of concept for database synchronization in mesh networks is performed using BATMAN-ADV with one of four proposed database synchronization strategies. The wireless links of the NerveNet mesh network are established using NerveNet's ERB. Data synchronization of the disaster response and monitoring platform is done using Hearsay and boxshare services from NerveNet. A cloud monitoring dashboard is designed to allow multiple regions multiple network monitoring. The performance and the feasibility of the network as well as the disaster response and monitoring framework as a whole is discussed in the next chapter.

# CHAPTER 4

# RESULTS AND DISCUSSION

## 4.1 Introduction

Figure 4.1 shows the prototype of the NerveNet testbed. The performance of the prototype is analysed from two perspectives, namely performance and database synchronization. Besides that, the feasibility of the disaster response and monitoring platform is analysed based on standard requirements.

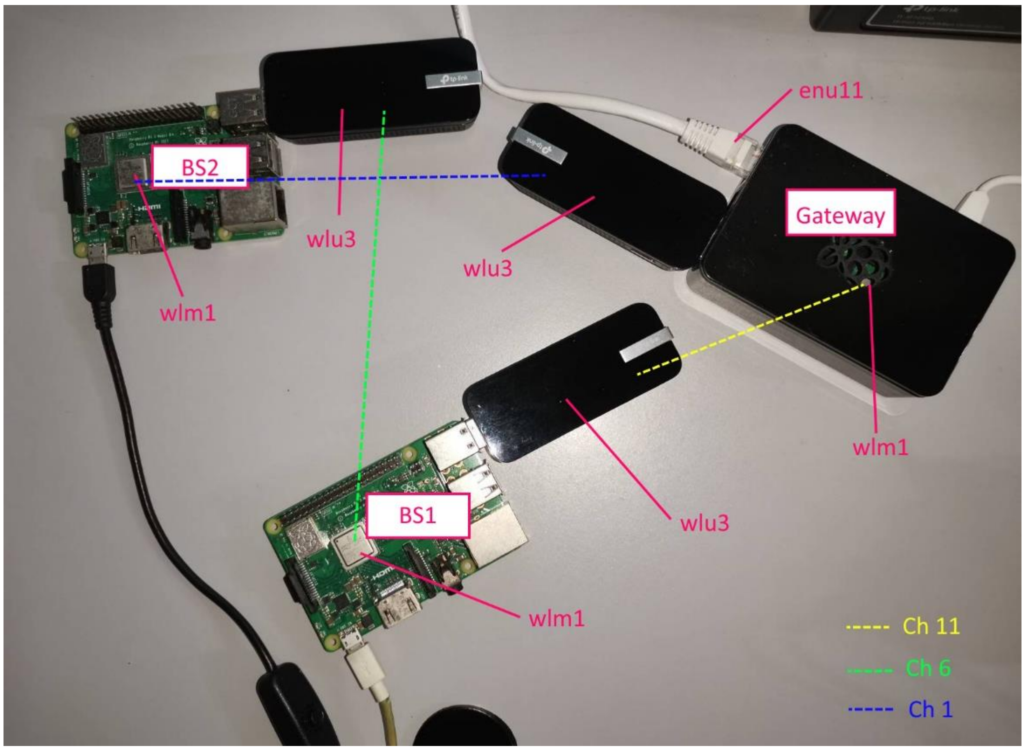


Figure 4.1: Prototype of the NerveNet Testbed.

To reduce electromagnetic interference between nodes, the nodes are distanced at least 1 meter from each other. The exact distances between each node are shown in Table 4.1.

Table 4.1: Distance Between Each Node During Network and Database Synchronization Benchmark.

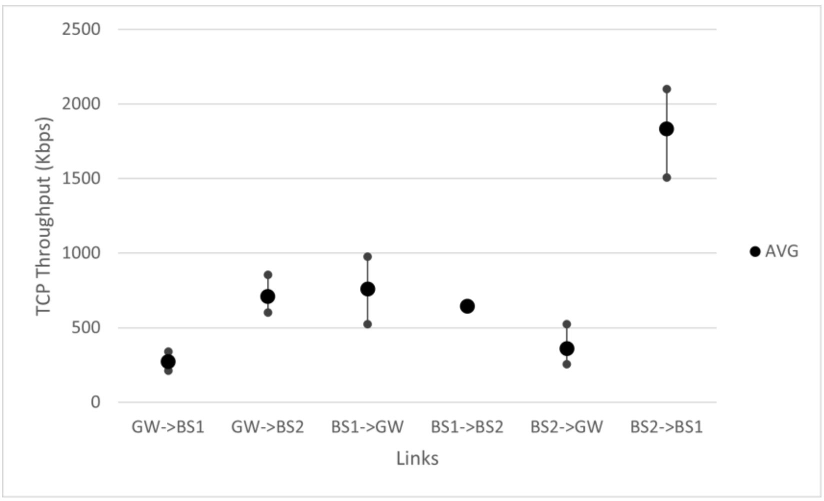
| **Node 1** | **Node 2** | **Distance (m)** |
|---|---|---|
| GW | BS1 | 1 |
| BS1 | BS2 | 2.1 |
| BS2 | GW | 2.35 |

## 4.2 Network Benchmark

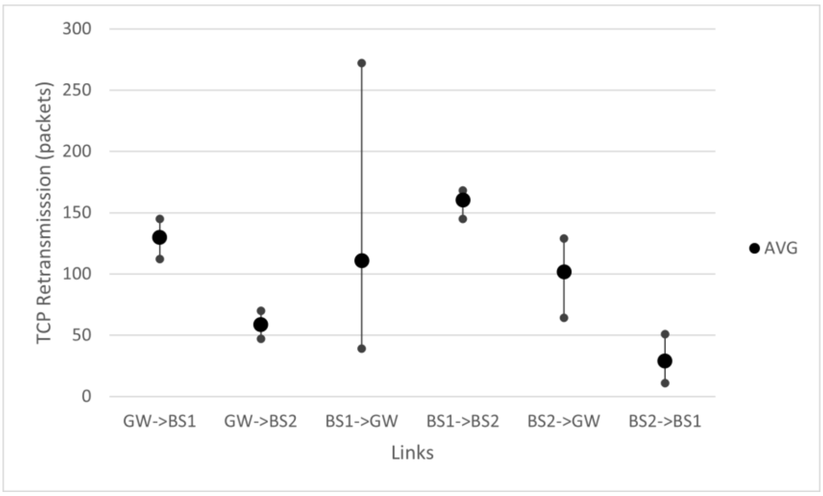
The network is benchmarked using iperf3 and ICMP ping. The network is benchmarked based on its UDP throughput, TCP throughput, TCP retransmission count, PDR, jitter and latency. The benchmark tests encompass all single-hop links and all two-hop links. The benchmark between single hop and two-hop links are compared.

### 4.2.1 Single Hop Benchmark

The TCP throughput and TCP retransmission count average-max-min chart for all single-hop links are shown in Figure 4.2.



(a)

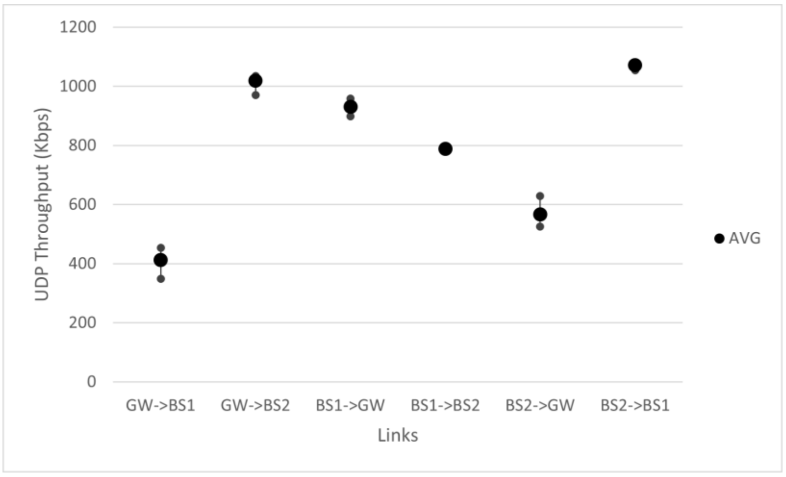


(b)

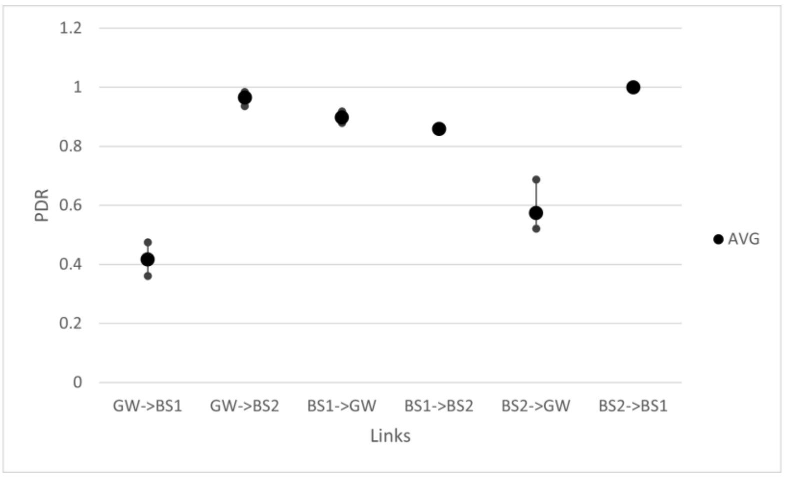
Figure 4.2: Average-Max-Min Chart of (a) the TCP Throughput and (b) TCP Retransmission Count for All Single Hop Links.

From Figure 4.2, communications between BS1 and BS2 have better TCP performance. This is because the links between BS1 and BS2 are less busy as compared to communications with the GW that is both the Internet Gateway and Network Manager of the network. Links that sourced from an STA (wlu3) sender to an AP (wlm1) receiver is also found to have a higher TCP throughput and lower TCP retransmission count. The plausible cause of such phenomena is that the STA utilizes a much performant wireless interface (wlu3) while the AP only utilizes the internal wireless interface of Raspberry Pi 3 Model B+ (wlm1) (Vouzis, 2018). These two conditions also explain why the link from BS2 to

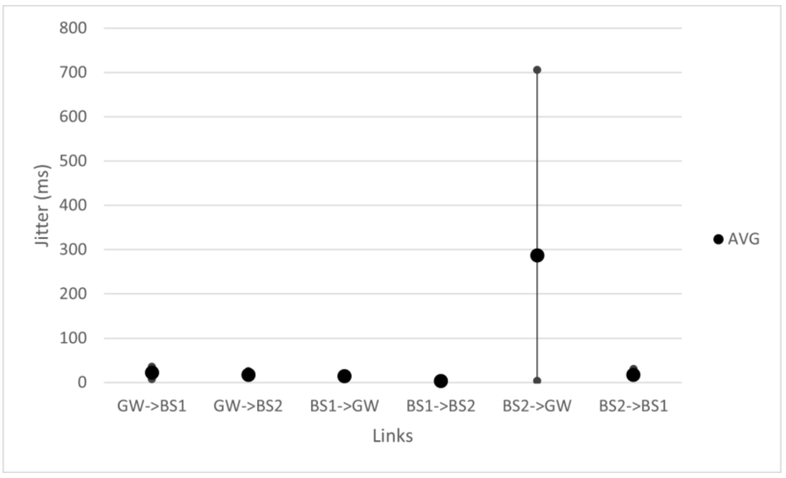
BS1 has the highest TCP throughput and lowest TCP retransmission from all links.



(a)



(b)



(c)

Figure 4.3: Average-Max-Min Chart of the (a) UDP Upload Throughput, (b) PDR and (c) Jitter for All Single Hop Links.

The performance result in Figure 4.3 is consistent with the TCP performance test results in Figure 4.2. From the effect of the higher network traffic in the links between GW with the other nodes and the connection direction of the links, a lower PDR value can be observed in the GW to BS1 and BS2 to GW links. The high jitter in the BS2 to GW link may be further justified by its distance between the nodes.

The latency of all single-hop links is measured using ICMP pings. The average, maximum and minimum latency of all single-hop links is shown in Figure 4.4.

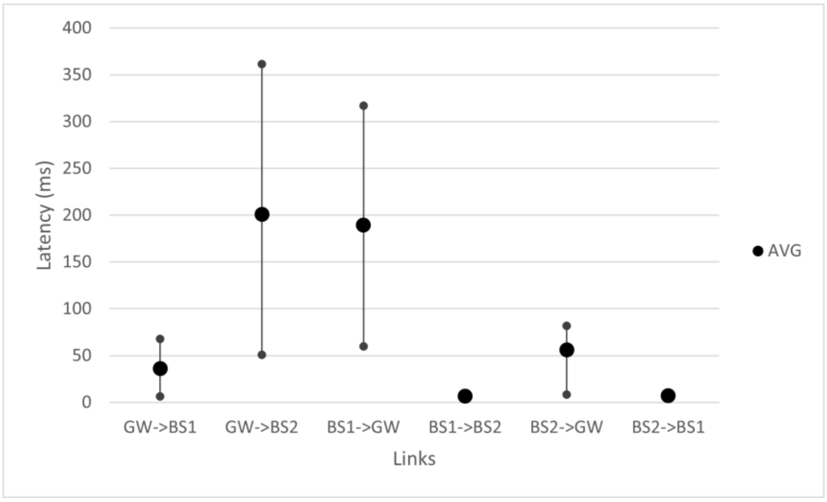
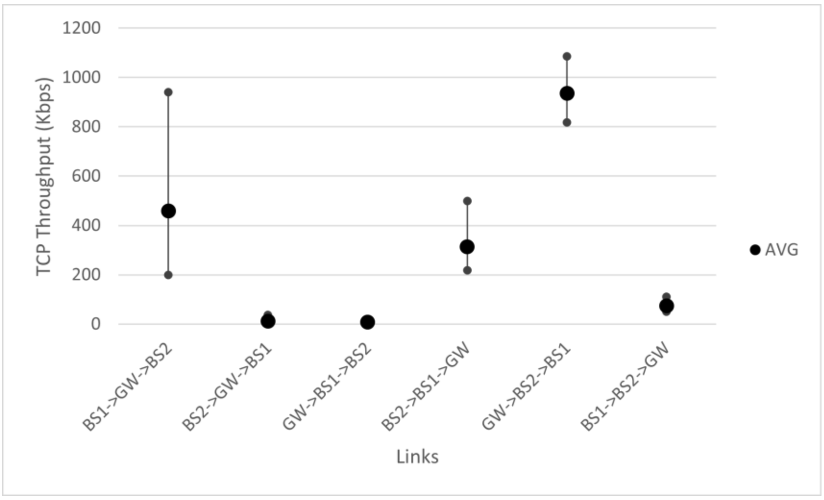


Figure 4.4: Average-Max-Min Chart of the Network Latency for All Single Hop Links.

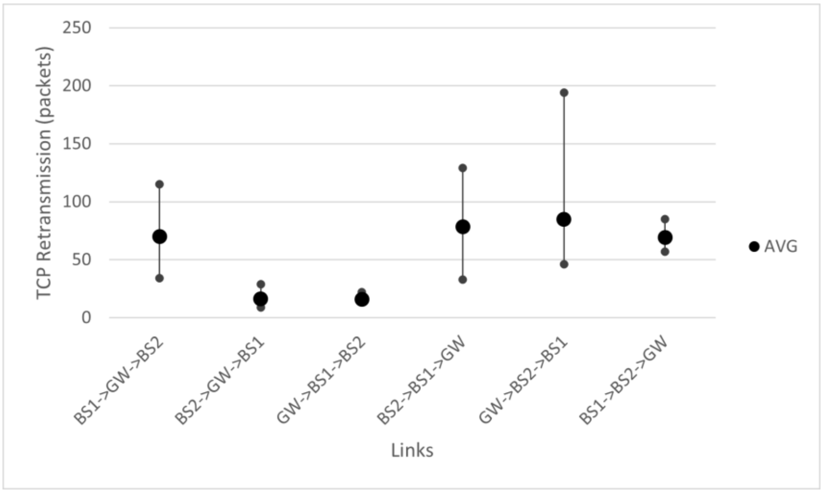
The average round trip latency of transmitting a 56 bytes payload using ICMP is at most 200 ms. Assume the link to be symmetric, the one-way latency is approximately 100 ms. This meets the requirement set for VoIP application networks by Cisco (Cisco Press, 2004).

### 4.2.2 Direct Link Against Two Hop Link

The TCP throughput and TCP retransmission count average-max-min chart for all measured two-hop links are shown in Figure 4.5. The UDP throughput, PDR and jitter average-max-min chart for all measured two-hop links are shown in Figure 4.6. The average, maximum and minimum latency of all measured two-hop links is shown in Figure 4.7.

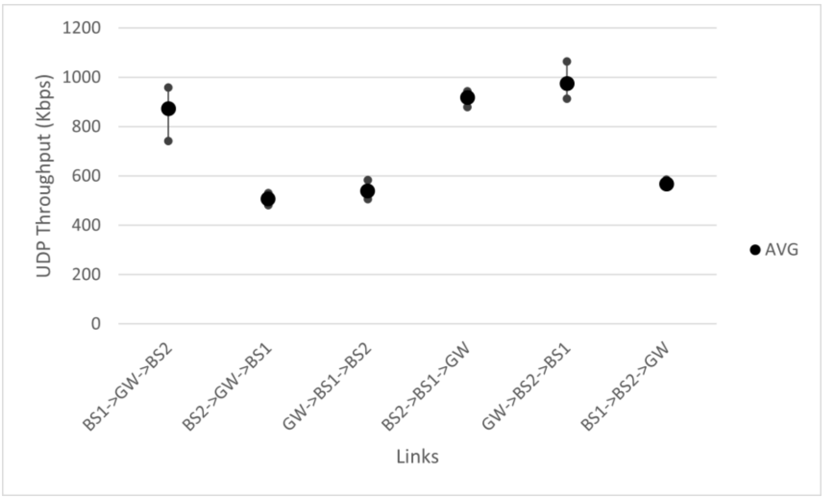


(a)

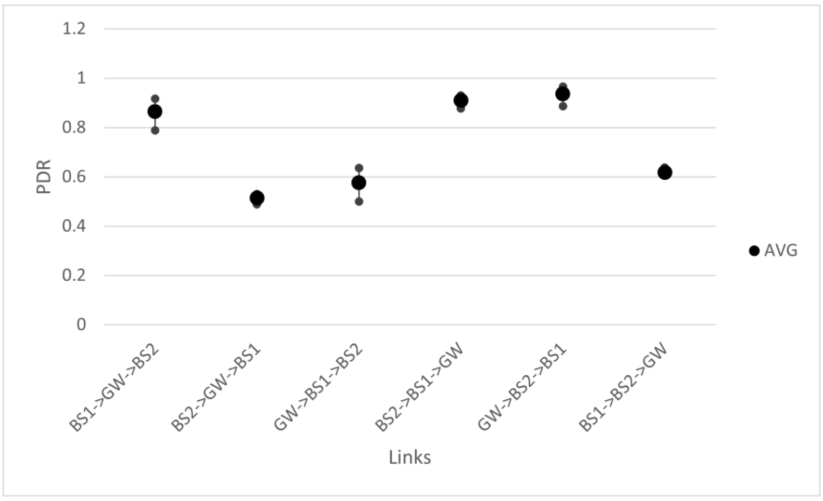


(b)

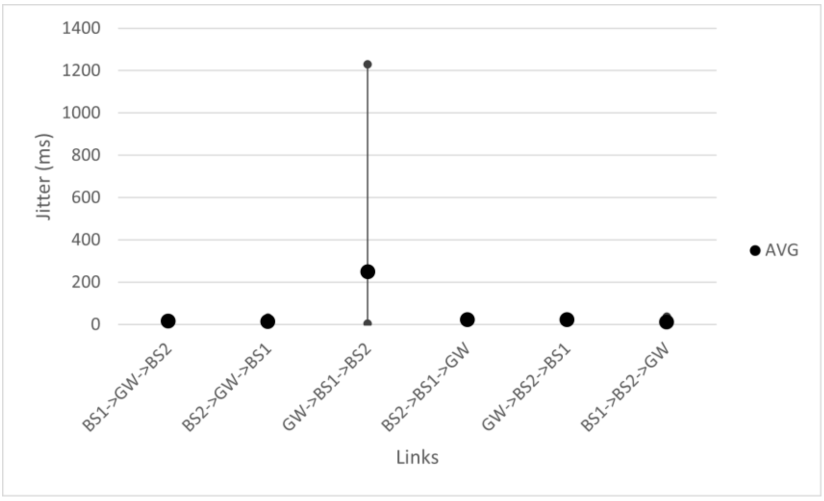
Figure 4.5: Average-Max-Min Chart of (a) the TCP Throughput and (b) TCP Retransmission count for All Measured Two Hop Links.



(a)



(b)



(c)

Figure 4.6: Average-Max-Min Chart of the (a) UDP Upload Throughput, (b) PDR and (c) Jitter for All Measured Two Hop Links.

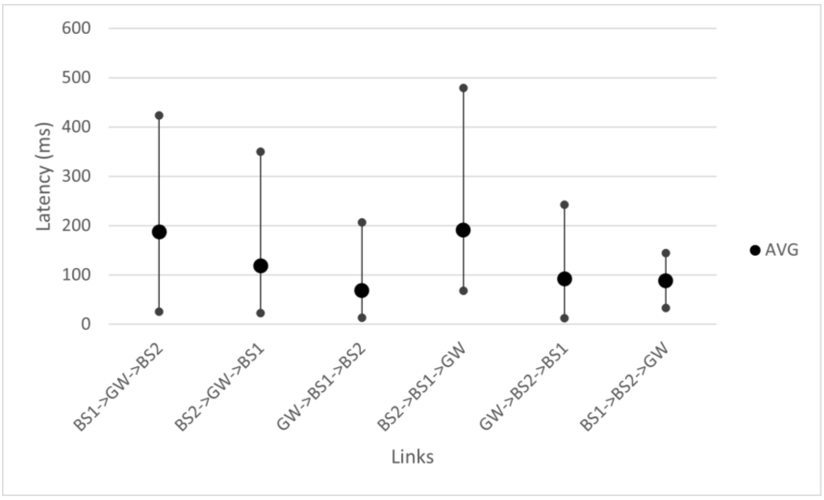


Figure 4.7: Average-Max-Min Chart of the Network Latency for All Measured Two Hop Links.

In a two-hop configuration, the effect of the connection direction is further amplified as the links are formed from two links with the same direction. The effect on a two-hop link is very obvious and can be seen in the measured TCP and UDP throughput performance. For example, the $BS2 \rightarrow GW \rightarrow BS1$, $GW \rightarrow BS1 \rightarrow BS2$ and $BS1 \rightarrow BS2 \rightarrow GW$ links consist of only AP to STA links. However, despite the bigger impact from the link directions, the TCP and UDP throughput of other links remain similar or even better to their single-hop counterpart. This is because the previously AP to STA single-hop link is now consisting of two separate STA to AP single-hop links which may have a better overall network performance. There is also no significant increase in TCP retransmission count and decrease in PDR between the two-hop configuration and its single-hop counterpart. The average latency of all two-hop links also remains below 200 ms.

## 4.3 Database Synchronization Benchmark

Four similar images with different sizes and resolution are used to measure the file synchronization latency of the Hearsay daemon. The image resolution and file sized is shown in Table 4.2. The image sent is shown in Figure 4.8 and the time taken for the images to be synchronized in each base station is shown in Figure 4.9.

Table 4.2: Image Resolution and File Size Used for Hearsay File Synchronization Performance Benchmarking.

| **Image File Name** | **Image Resolution** | **File Size** |
|---|---|---|
| img_small.jpg | 640 × 360 px | 108 KB |
| img_mid.jpg | 1920 × 1080 px | 950 KB |
| img_large.jpg | 2400 × 1350 px | 1.41 MB |
| img_xl.jpg | 7679 × 4320 px | 10.6 MB |

Figure 4.8: The Image Used for Synchronization (img_small.jpg).

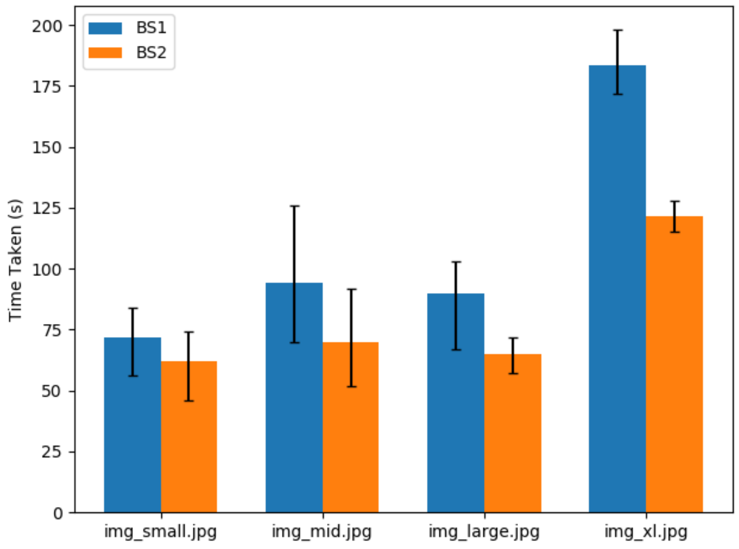


Figure 4.9: Time Taken for Different Sized Images to Be Synchronized from GW to BS1 and BS2 Respectively.

From Figure 4.9, the time taken for each image to be synchronized in both BS1 and BS2 increases as the size of the file increases. Based on the test results, the maximum time taken for an image with a file size equal to or lesser than 1.4 MB in a three nodes configuration requires at most 2 minutes and 6 seconds for it to be fully synchronized depending on the network condition. However, as the system only requires transmitting low-resolution images of the scene, this is sufficient for disaster response and monitoring platform. The estimated database synchronization throughput for GW to BS1 and GW to BS2 can be by calculating the average number of bits synchronized in a second. The average, maximum and minimum database synchronization throughput for GW to BS1 and GW to BS2 are shown in Table 4.3.

Table 4.3: Average, Maximum and Minimum Estimated Database Synchronization Throughput.

| **Source** | **Destination** | **Maximum (Kbps)** | **Average (Kbps)** | **Minimum (Kbps)** |
|---|---|---|---|---|
| GW | BS1 | 473.6552 | 173.8300 | 12.0000 |
| GW | BS2 | 1154.2659 | 498.2454 | 13.9355 |

From Table 4.3, the average throughput of the data synchronization is comparably lower than the actual TCP and UDP throughput of the network. This is due to an extra delay introduced by the synchronization polling duration to verify the checksum of each table between the source and synchronized node. This reduces the based bandwidth used for database synchronization and only synchronize the database when needed. As the link between GW and BS1 is an AP to STA link while the link between GW and BS2 is an STA to AP link, the overall synchronization performance of BS1 is significantly weaker compared to that of BS2.

## 4.4 Platform Design Feasibility

In the year 2014, the International Telecommunication Union has released a standard regarding the requirements for network resilience and recovery. Annex B of the standard lists the requirements for improvement of network resilience and recovery using local wireless mesh network based on de-centralized mesh architecture. The compliance of the disaster response and monitoring platform is discussed to ensure its feasibility and capability as a disaster resilient system.

### 4.4.1 Annex B.5: Network Requirements

Based on Annex B.5, the network must be able to resume its communication using surviving nodes when parts of the network are damaged or overloaded. This holds for the platform as it uses a mesh topology and is able to send network packets through other paths even if its original path is temporary or permanently interrupted. The platform also complies with the standard in terms of mobility, address resolution and multicast feature even in an event of connection interruption to the Internet. NerveNet is also able to re-route packets and restore the network by its Network Manager even if a majority of the network is damaged (Inoue and Owada, 2017).

### 4.4.2 Annex B.6: Portable Radio Relay Node

This annex requires the relay nodes of the network to be portable. Antennas for both connections between relay nodes and terminal devices are required. The relay nodes must also be physically portable. As NerveNet can be deployed in microcomputers such as Raspberry Pi and Intel NUC, it is portable. NerveNet also supports connections for both relay nodes and terminal devices. Thus, the platform complies with the requirements in Annex B.6 by using NerveNet as its local mesh network.

### 4.4.3 Annex B.10: Service Platform Requirements in the Local Private Network

Annex B.10 provides the requirements of the software services within the local private network of the platform. The testbed includes an example application of a constant human counting system, it can provide the first level of emergency response information. Additional applications can also be developed to provide refuge instruction, safety confirmation and relief request features as required in the standard. All applications can utilize the Hearsay daemon provided by

NerveNet to have a synchronized and distributed database in multiple nodes and/or terminals before, during or after a disaster.

### 4.4.4 Annex B.7, B.8 and B.9

Annex B.7 states the requirement of aerial nodes that used unmanned aerial vehicle (UAV) to provide connectivity through the platform in isolated locations. Annex B.8 also states that some gateway node should be connected through a satellite connection to provide a more reliable connection to the wide-area network. Annex B.9 further require the network to have gateway nodes that utilize satellite connection to be mounted on vehicles to provide higher mobility. Fortunately, NerveNet has supports for such connections as stated in their 2017 paper titled “NerveNet Architecture and Its Pilot Test in Shirahama for Resilient Social infrastructure”.

## 4.5 Summary

From the study, the testbed deployed is considered to have an acceptable network performance for a disaster response and monitoring network. The testbed has similar network performance for its two-hop links as compared to their single-hop counterparts. Furthermore, the disaster response and monitoring platform proposed can meet all the requirements laid out by the International Telecommunication Union for improving network resilience and recovery using a local wireless mesh network.

# CHAPTER 5

# CONCLUSION AND RECOMMENDATIONS

## 5.1 Conclusion

In conclusion, a testbed for disaster response and monitoring platform using NerveNet has been designed and deployed. The tools such as the Hearsay daemon provided in NerveNet has been proven beneficial for application services within the regional local private network. The platform design meets the requirements set by the International Telecommunication Union. The performance of the platform is also within an acceptable range of a regional disaster response and monitoring network. The outcome of the real implementation can serve as a guideline on designing and deploying a disaster response and monitoring platform using NerveNet. It will promote disaster-resilient telecommunications and distributed application development for disaster response.

## 5.2 Recommendations for Future Work

Due to time limitation, the network performance analysis has been done on a NerveNet deployment with unsolved network issues. To have a better and accurate performance study, the issues should first be solved before conducting the performance analysis in the network. A better and extensive network performance test should be conducted to have a better understanding of the feasibility of the platform.

Besides, a larger testbed can be deployed by increasing the total NerveNet nodes count. Terminal nodes can also be included in the platform testbed to provide a simpler connection method for sensors, users and devices. Additional disaster response and monitoring application and features can be developed and tested using this testbed.

Furthermore, LoRa NerveNet deployments can be incorporated into the platform to provide long-distance communication between NerveNet nodes. The NerveNet deployment can also utilize the latest Docker container deployment method rather than manual image write to have a much more efficient deployment process. Last but not least, improvement in NerveNet user documentation is needed to further promote and simplify the process of NerveNet deployments.

# REFERENCES

Abdalla, T., Rey-Moreno, C., Tucker, W.D. and Bagula, A., 2015. Clustered Multi-layer Multi-protocol Wireless Mesh Networks. *Southern Africa Telecommunication Networks & Applications Conference*, I, pp.99–104.

Anon 2021. *Layers of OSI Model Explained*. [online] Available at: <https://www.guru99.com/layers-of-osi-model.html> [Accessed 5 Apr. 2021].

Anon 2021. *SQL Vs NoSQL Exact Differences And Know When To Use NoSQL And SQL*. [online] Available at: <https://www.softwaretestinghelp.com/sql-vs-nosql/#:~:text=NoSQL Cons%3A,-The benefits of&text=NoSQL offers only eventual consistency,the case of NoSQL databases.> [Accessed 2 Apr. 2021].

Arora, S., 2021. *Webinar Wrap-up: Edge Computing Vs. Cloud Computing*. [online] Available at: <https://www.simplilearn.com/edge-computing-vs-cloud-computing-article#:~:text=Edge computing is used to,connectivity to a centralized location.> [Accessed 5 Apr. 2021].

Arya, S., 2017. *What the hell is scalable code anyway?* [online] Available at: <https://blog.sarasarya.com/what-the-hell-is-scalable-code-anyway-f6626ad78227> [Accessed 5 Apr. 2021].

Barnhill, B. and David, M., 2021. *Row vs Column Oriented Databases*. [online] Available at: <https://dataschool.com/data-modeling-101/row-vs-column-oriented-databases/> [Accessed 2 Apr. 2021].

Čandrlić, G., 2015. *High Latency vs Low Bandwidth - Impact on Web Performance*. [online] Available at: <https://www.globaldots.com/blog/high-latency-vs-low-bandwidth-impact-web-performance#:~:text=Latency is also referred to,measured in milliseconds (ms).&text=Excessive latency creates bottlenecks that,pipe%2C thus decreasing effective bandwidth.> [Accessed 4 Apr. 2021].

Cisco Press, 2004. *Quality of Service Design Overview*. [online] Available at: <https://www.ciscopress.com/articles/article.asp?p=357102#:~:text=Average one-way jitter should,and Layer 2 media overhead).> [Accessed 10 Apr. 2021].

Concepta, 2019. *The Importance of Scalability In Software Design*. [online] Available at: <https://www.conceptatech.com/blog/importance-of-scalability-in-software-design> [Accessed 5 Apr. 2021].

Cox, J., 2020. *What is Network Throughput and How to Measure & Monitor it!*

[online] Available at: <https://www.ittsystems.com/network-throughput/> [Accessed 4 Apr. 2021].

Dahmouni, H., Girard, A. and Sansò, B., 2012. An analytical model for jitter in IP networks. *Annales des Telecommunications/Annals of Telecommunications*, 67(1–2), pp.81–90.

DoCoMo, N., 2011. Maintaining Communications Capabilities during Major Natural Disasters and other Emergency Situations Final Report.

Drake, M., 2019. *A Comparison of NoSQL Database Management Systems and Models*. [online] Available at: <https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models> [Accessed 2 Apr. 2021].

Inoue, M. and Owada, Y., 2017. NerveNet architecture and its pilot test in Shirahama for resilient social infrastructure. *IEICE Transactions on Communications*, E100B(9), pp.1526–1537.

International Telecomminication Union, 2014. *Requirements of Network Resilience and Recovery*. 1st ed. [online] International Telecommunication Union. Available at: <https://www.itu.int/en/ITU-T/focusgroups/drnrr/Documents/fg-drnrr-tech-rep-2014-6-NRR-requirement.pdf>.

Jindal, R., 2016. REVIEW PAPER ON DATABASE SYNCHRONIZATION BETWEEN LOCAL AND. *International Journal of Engineering Sciences and Research Technology*, 5(7), pp.1396–1400.

Khan, M.K.U. and Ramesh, K.S., 2019. Effect on Packet Delivery Ratio (PDR) & Throughput in Wireless Sensor Networks Due to Black Hole Attack. *International Journal of Innovative Technology and Exploring Engineering*, [online] 8(12S), pp.428–432. Available at: <https://www.ijitee.org/wp-content/uploads/papers/v8i12S/L110710812S19.pdf>.

Manandhar, G., 2020. *6 dev and operations factors to consider for software scalability to meet high demands*. [online] Available at: <https://geshan.com.np/blog/2020/12/software-scalability/#:~:text=Software scalability is an attribute,%2C overhauls%2C and resource reduction.> [Accessed 5 Apr. 2021].

Marr, B., 2017. *What Is Digital Twin Technology - And Why Is It So Important?* [online] Available at: <https://www.forbes.com/sites/bernardmarr/2017/03/06/what-is-digital-twin-technology-and-why-is-it-so-important/?sh=40226fc62e2a> [Accessed 5 Apr. 2021].

Nath, K., 2019. *A guide to understanding database scaling patterns*. [online] Available at: <https://www.freecodecamp.org/news/understanding-database-scaling-patterns/>.

Owada, Y., Inoue, M. and Ohnishi, M., 2011. Regional wireless network platform for context-aware services and its implementation. *Proceedings - 2011 10th International Symposium on Autonomous Decentralized Systems, ISADS 2011*, pp.539–544.

Owada, Y., Sato, G., Temma, K., Kuri, T., Inoue, M. and Nagano, T., 2019. An Implementation of Layer 2 Overlay Mesh Network and Edge Computing Platform for IoT. *2019 12th International Conference on Mobile Computing and Ubiquitous Network, ICMU 2019*.

Pinto, R., Lu, D., Rodrigues, E.T. and Oliveira, D.L., 2010. WMM - Wireless Mesh Monitoring (midterm report).

Sandberg, B., 2015. *Networking: The Complete Reference*. 3rd ed. McGraw-Hill Education.

Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V., 2003. *RTP: A transport protocol for real-time applications*. [online] *RFC Editor*. Available at: <https://www.rfc-editor.org/info/rfc3550>.

Shodiq, M., Wongso, R., Pratama, R.S., Rhenardo, E. and Kevin, 2015. Implementation of Database Synchronization with Data Marker using Web Service Data. *International Conference on Computer Science and Computational Intelligence*, 59, pp.366–372.

Smith, P., Fessi, A., München, T.U., Lac, C., Telecom, F. and Labs, O., 2011. Network Resilience : A Systematic Approach. (July).

Solid IT, 2021. *DB-Engines Ranking of Graph DBMS*. [online] Available at: <https://db-engines.com/en/ranking/graph+dbms> [Accessed 2 Apr. 2021].

Sterbenz, J.P.G., Hutchison, D., Çetinkaya, E.K., Jabbar, A., Rohrer, J.P., Schöller, M. and Smith, P., 2010. Resilience and survivability in communication networks : Strategies , principles , and survey of disciplines. *Computer Networks*, [online] 54(8), pp.1245–1265. Available at: <http://dx.doi.org/10.1016/j.comnet.2010.03.005>.

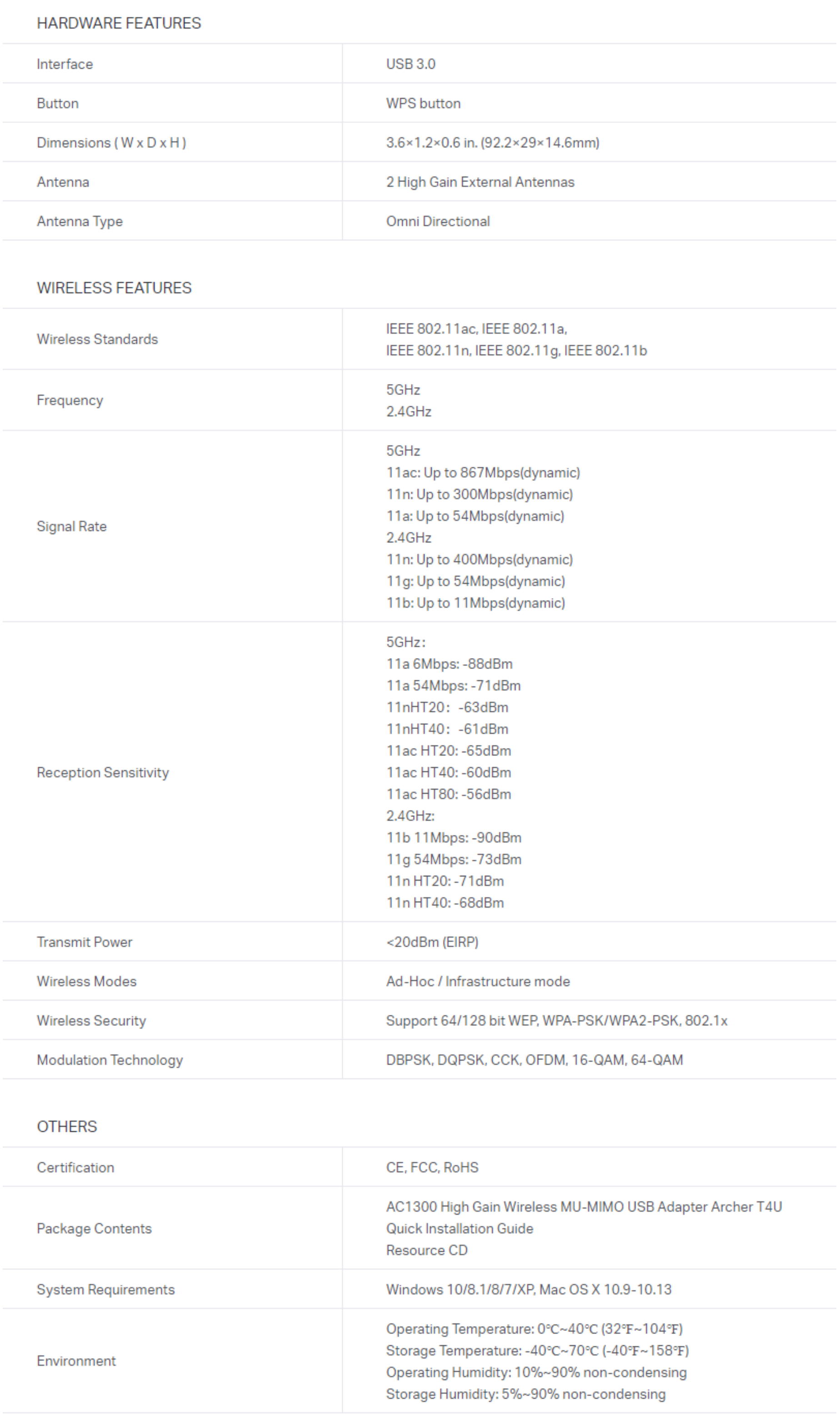
Vázquez, F., 2019. *Graph Databases. What's the Big Deal?* [online] Available at: <https://towardsdatascience.com/graph-databases-whats-the-big-deal-ec310b1bc0ed> [Accessed 2 Apr. 2021].

Vouzis, P., 2018. *Raspberry Pi 3B+ iPerf WiFi Performance*. [online] Available at: <https://netbeez.net/blog/raspberry-pi-3b-iperf-wifi-performance/#:~:text=The 3B%2B came with a,802.11n at 2.4GHz).> [Accessed 14 Apr. 2021].

YML, 2020. *Native VS Hybrid Mobile Apps — Here’s How To Choose*. [online] Available at: <https://uxplanet.org/native-vs-hybrid-mobile-apps-heres-how-to-choose-192ecbf04da8> [Accessed 5 Apr. 2021].

# APPENDICES

## APPENDIX A: TP-Link AC1300 Archer T4U USB Adapter Specification



| | |
|---|---|
| HARDWARE FEATURES | |
| Interface | USB 3.0 |
| Button | WPS button |
| Dimensions ( W x D x H ) | 3.6×1.2×0.6 in. (92.2×29×14.6mm) |
| Antenna | 2 High Gain External Antennas |
| Antenna Type | Omni Directional |
| WIRELESS FEATURES | |
| Wireless Standards | IEEE 802.11ac, IEEE 802.11a,<br>IEEE 802.11n, IEEE 802.11g, IEEE 802.11b |
| Frequency | 5GHz<br>2.4GHz |
| Signal Rate | 5GHz<br>11ac: Up to 867Mbps(dynamic)<br>11n: Up to 300Mbps(dynamic)<br>11a: Up to 54Mbps(dynamic)<br>2.4GHz<br>11n: Up to 400Mbps(dynamic)<br>11g: Up to 54Mbps(dynamic)<br>11b: Up to 11Mbps(dynamic) |
| Reception Sensitivity | 5GHz：<br>11a 6Mbps: -88dBm<br>11a 54Mbps: -71dBm<br>11nHT20：-63dBm<br>11nHT40：-61dBm<br>11ac HT20: -65dBm<br>11ac HT40: -60dBm<br>11ac HT80: -56dBm<br>2.4GHz:<br>11b 11Mbps: -90dBm<br>11g 54Mbps: -73dBm<br>11n HT20: -71dBm<br>11n HT40: -68dBm |
| Transmit Power | <20dBm (EIRP) |
| Wireless Modes | Ad-Hoc / Infrastructure mode |
| Wireless Security | Support 64/128 bit WEP, WPA-PSK/WPA2-PSK, 802.1x |
| Modulation Technology | DBPSK, DQPSK, CCK, OFDM, 16-QAM, 64-QAM |
| OTHERS | |
| Certification | CE, FCC, RoHS |
| Package Contents | AC1300 High Gain Wireless MU-MIMO USB Adapter Archer T4U<br>Quick Installation Guide<br>Resource CD |
| System Requirements | Windows 10/8.1/8/7/XP, Mac OS X 10.9-10.13 |
| Environment | Operating Temperature: 0℃~40℃ (32℉~104℉)<br>Storage Temperature: -40℃~70℃ (-40℉~158℉)<br>Operating Humidity: 10%~90% non-condensing<br>Storage Humidity: 5%~90% non-condensing |